université
PARIS-SACLAY

Inference of graph transformation rules for the design of geometric modeling operations

Inférence de règles de transformations de graphe pour la conception d'opérations de modélisation géométrique

Thèse de doctorat de l'université Paris-Saclay

École doctorale nº 573, Interfaces : matériaux, systèmes, usages
Spécialité de doctorat : Informatique
Graduate School : Sciences de l'ingénierie et des systèmes
Référent : CentraleSupélec

Thèse préparée dans l'unité de recherche : Mathématiques et Informatique pour la Complexité et les Systèmes (Université Paris-Saclay, CentraleSupélec) sous la direction de **Pascale LE GALL**, Professeur, et le co-encadrement de **Hakim BELHAOUARI**, Maître de conférence (Université de Poitiers), et **Agnès ARNOULD**, Maître de conférence (Université de Poitiers).

Thèse soutenue à Paris-Saclay, le 29 novembre 2022, par

Romain Pascual

Composition du jury :

Membres du jury avec voix délibérative

Céline HUDELOT, Professeur, CentraleSupélec	Présidente
Guillaume DAMIAND, Directeur de recherche, CNRS, Université Claude Bernard	Rapporteur et Examinateur
Reiko HECKEL, Professeur, University of Leicester	Rapporteur et Examinateur
Nicolas BEHR, Chargé de recherche, CNRS, Université Paris Cité	Examinateur
Bedrich BENES, Professeur, University of Purdue	Examinateur
Jean-Luc MARI, Professeur, Université Aix-Marseille	Examinateur

Thèse de doctorat

NNT: 2022UPAST146

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre : Inférence de règles de transformations de graphe pour la conception d'opérations de modélisation géométrique

Mots clés : transformations de graphes, préservation de consistance, extension de la réécriture par double-pushout, modélisation géométrique à base topologique, cartes combinatoires, inférence d'opérations .

Résumé : Dans cette thèse, nous proposons une formalisation des opérations de modélisation géométrique comme des règles de transformation de graphes.

Dans une première partie, nous étudions la conception d'un langage dédié à base de règles. Nous décrivons les modèles combinatoires des cartes généralisées et orientées comme des graphes étiquetés, assujettis à des conditions de cohérence. À cette description topologique, s'ajoute une gestion de la géométrie à l'aide d'attributs. Cette formalisation permet l'étude des opérations de modélisation comme des règles de réécritures de graphes. Cette étude des règles couvre deux aspects : la préservation de la consistance du modèle et la généricité des opérations décrites. La préservation de la consistance est la motivation première pour l'utilisation des transformations de graphes afin de garantir qu'une opération produit des objets bien formés. Nous assurons la préservation de la consistance topologique et géométrique par le biais de conditions syntaxiques vérifiées statiquement sur les règles. Il convient aussi de s'assurer qu'une règle de réécriture est en adéquation avec les opérations usuelles manipulées en modélisation géométrique. En particulier, puisqu'une règle décrit exactement une transformation, nous étendons les règles en schémas de règles afin d'abstraire la topologie sous-jacente. Nous présentons une extension semi-globale de la réécriture usuelle par DPO en incorporant un produit de graphes simulant l'application d'une fonction de renommage.

Dans une seconde partie nous présentons un mécanisme d'inférence d'opérations. Partant du principe qu'une opération peut-être simplement décrite à partir d'un croquis ou d'un exemple, nous proposons de reconstruire des opérations à partir d'un exemple représentatif constitué d'un objet de départ et de l'objet cible. Le mécanisme d'inférence exploite la régularité des cartes généralisées et du langage dédié précédemment défini. Plus précisément, nous envisageons la question de l'inférence d'opérations topologiques comme la construction inverse de la spécialisation d'un schéma de règle vers une opération. La question de l'inférence des modifications géométriques pourrait admettre de multiples solutions, étant donné le type de géométrie et la nature des modifications appliquées. Ici, nous proposons de traiter des transformations affines de valeurs évoluant dans un espace vectoriel que l'on résout comme un problème de satisfaction de contraintes. Nous avons implémenté le mécanisme d'inférence dans Jerboa, une plateforme de conception de modeleurs. La première partie de cette thèse permet ainsi de construire un cadre formel qui est de facto masqué à l'utilisateur, mais demeure nécessaire pour la conception d'opérations de modélisations géométriques via notre mécanisme d'inférence.

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Title: Inference of graph transformation rules for the design of geometric modeling operations

Keywords: graph transformation, consistency preservation, extensions to double-pushout rewriting, topology-based geometric modeling, combinatorial maps, inference of operations .

Abstract: In this thesis, we present a formalization of geometric modeling operations as rules from the theory of graph transformation.

First, we investigate the construction of a dedicated rule-based language. We describe the combinatorial models of generalized and oriented maps as labeled graphs, subject to consistency conditions. This topological representation is coupled with a description of the geometry using attributes. This formalization allows the study of modeling operations such as graph rewriting rules. The rules analysis covers two aspects: the preservation of the model consistency and the genericity of the described operations. Consistency preservation is the primary incentive for using graph transformations. Indeed, modifications of a well-formed object should result in a well-formed object. We ensure the preservation of topological and geometric consistency through syntactic conditions statically checked on the rules. In addition, we ensure that rewriting rules meet the requirements to describe the usual operations used in geometric modeling. In particular, since a rule fully describes a transformation, we extend the rules into rule schemes in order to abstract over the underlying topology. We present a semi-global extension of the usual DPO rewriting by incorporating a graph product simulating the application of a relabeling function.

Secondly, we present a mechanism for the inference of operations. Given that an operation can be simply described from a sketch or an example, we propose to reconstruct operations from a representative example made of an initial and a target object. The inference mechanism exploits the regularity of generalized maps and the dedicated language previously. More precisely, we consider the process of inferring topological operations as the inverse construction of the specialization of a rule scheme into an operation. The inference of geometric modifications could admit multiple solutions, given the type of data and the nature of the modifications applied. Here, we propose to consider affine transformations of values from a vector space, which we solve as a constraint satisfaction problem. We have implemented the inference mechanism in Jerboa, a platform for the design of geometric modelers. The first part of this thesis, therefore, allows building a formal framework that is de facto hidden from the user but is still necessary for the conception of geometric modeling operations via our inference mechanism.

À Alain et Élios, j'espère que vous auriez été fiers

Remerciements

Le présent manuscrit contient, somme toute, un résumé assez fidèle des pérégrinations (scientifiques bien sûr !) que j'ai pu réaliser lors de ces trois années qui ont constitué ma thèse. Quand bien même l'usage oblige que mon nom figure en grand sur la page de couverture de ce document, nombreux sont ceux sans qui vous ne seriez pas en train de lire ce document. Cette thèse, c'est autant la mienne que la leur.

Je tiens à préciser que certains seront remerciés plusieurs fois (sûrement qu'ils l'ont mérité), et que d'autres seront peut-être oubliés. Si vous lisez ce document, que je vous ai oublié et que vous vous sentez lésé, alors je vous remercie chaleureusement[1].

Je suis entré à l'École Centrale Paris en septembre 2015. Bien que peu motivé à l'idée d'étudier le dimensionnement des ailettes, et découvrant avec incrédulité qu'il suffisait de jeter des œufs par la fenêtre pour être un bon ingénieur, deux points m'ont fasciné lors de mon entrée : la présence d'un terrain de rugby au milieu du campus et la possibilité de mener une activité de recherche en parallèle des études (merci Papa d'avoir lu la brochure de l'école). C'est ainsi que le lundi 28 septembre 2015 à 11:30, j'ai toqué à la porte de celle qui deviendra par la suite ma directrice de thèse, Pascale Le Gall. Cela fait donc maintenant près de sept ans que nous nous connaissons Pascale, merci pour ta générosité, ton enthousiasme sans bornes et tes conseils pour naviguer au mieux dans cette jungle qu'est le milieu de la recherche, si le meilleur conseil que l'on peut donner à un futur doctorant est de bien choisir ses encadrants de thèse alors, je crois que je n'aurais pas pu mieux choisir. C'est aussi par le biais du parcours recherche que je rencontre Thomas Bellet, ancien doctorant de Pascale, et, à l'époque, postdoc au MAS. C'est grâce à Thomas que je découvre Java, la modélisation 3D et les doubles sommes amalgamées pour la réécriture de graphes, mais ce que je découvre avant tout, c'est un encadrant passionné et désireux de partager sa passion. Si j'ai plongé (sombré ?) dans Jerboa, c'est grâce à toi et je t'en remercie.

Lors de ma deuxième année de parcours recherche, j'ai eu la chance de faire un séjour de deux jours au laboratoire XLIM, à l'université de Poitiers. C'est ainsi que je rencontre Hakim Belhaouri, Agnès Arnould et Valentin Gauthier (alors doctorant avec Hakim et Agnès), le 21 novembre 2016. Peut-être que je ne le réalisais pas alors, mais je rencontrais mes futurs co-encadrants de thèse. Une paire de lunettes derrière une rangée d'écran installée sur un bureau partagé avec un Sangoku en perle à repasser, et un grand sourire sur le visage jovial de celle qui m'ouvre la porte, un cliché que je ne suis pas près d'oublier. À Hakim, pour cette double vie partagée tels Alfred et Bruce Wayne qui nous auront permis d'avancer presque plus vite de nuit que de jour, pour ton introduction dans la communauté de modélisation géométrique, pour ces commentaires échangés sur les orateurs que nous avons pu voir à Dijon ou à Reims, un immense merci. À Agnès, pour ta bonne

[1] et espère que vous pardonnerez mon étourderie !

humeur, ton franc-parler, tes désaccords scientifiques qui m'ont permis de prendre une certaine forme de recul sur de nombreuses questions, si tu représentes le futur d'un doctorat supervisé avec Pascale, je ne me fais pas trop de souci pour mon avenir.

Si vous lisez actuellement ce document, c'est aussi parce que ma thèse est aujourd'hui terminée. Je l'ai présentée et défendue il y a maintenant presque deux mois, le 29 novembre 2022. Cette présentation n'aurait eu de sens sans les six membres de mon jury, que je souhaite remercier chaleureusement. Tout d'abord, merci à Céline Hudelot de m'honorer d'avoir accepté de présider mon jury, de t'être rendue disponible une demi-journée pour m'écouter, mais aussi pour les conseils et discussions que nous avons pu avoir au cours de mes années au MAS puis au MICS. Je remercie mes rapporteurs, Guillaume Damiand et Reiko Heckel, pour la relecture de mon manuscrit. Vous n'avez été effrayés ni par la longueur, ni la pluridisciplinarité de ma recherche et je vous suis profondément reconnaissant de votre analyse de mon travail. Je vous ai chacun rencontré en ligne (Covid oblige), aux journées du GTMG 2020 et au séminaire GReTA, avant de vous rencontrer en chair et en os, à ICGT et aux journées du GTMG 2022, et j'espère que nous aurons l'occasion de nous recroiser à nouveau. À Nicolas Behr pour ta bienveillance lors de nos rencontres, pour ton introduction dans la communauté des transformations de graphes. À Bedrich Benes, tu as été mon tuteur de stage de M2, je me souviens de ta patience lorsque j'abordais les questions que tu me proposais exactement selon l'angle qui ne convenait pas. À Jean-Luc Mari, rencontré à Dijon alors qu'Hakim portait un t-shirt à l'effigie de ton groupe de musique, cela ne pouvait être qu'un bon présage. Tant d'experts dont j'ai pu lire les travaux, parfois avec trop d'admiration, qui ont examiné les miens, j'en reste sans mots.

Je tiens aussi à remercier celles et ceux sans qui mon travail n'aurait simplement pas pu se faire. À Suzanne Thuron, toujours de bonne humeur et prompte à résoudre tous les problèmes administratifs de l'Ecole Doctorale. À Fabienne Brosse, faut-il dire secrétaire du laboratoire ou superwoman ? À Dany Kouoh Etamè, toujours jovial. Au staff de l'Academic Writing Center, en particulier Melissa Ann Thomas pour vos relectures, Calvin Peck pour votre aide à préparer la soutenance et surtout Divya Madhavan pour m'avoir appris que l'on peut écrire un article scientifique sans pour autant effacer toute idée de style littéraire. À Delphine Le Piolet, Quentin Touzé et Stéphanie Raynaud du pôle IST pour leurs réponses à toutes mes questions sur les modalités d'écriture et de publication du présent manuscrit.

Je tiens aussi à remercier l'ensemble des personnes égarées dans un recoin du bâtiment Bouygues, plus communément appelé laboratoire MICS, bien que la proximité du plateau de Saclay avec toute forme de civilisation puisse porter à interrogation sur la sanité des dites personnes. À Anaëlle, Céline, Myriam et Wassila pour vos discussions autour d'un déjeuner, d'un café ou dans un RER B arrêté quelque part entre Massy et Bourg-la-Reine. Aux indétronables du laboratoire, Érick, Ludovic et Vincent, vous étiez là par temps de pluie, de neige ou de Covid j'ai particulièrement apprécié nos discussions au fil des années. À Marc, j'espère que les étudiants finiront par comprendre que "trivial" n'est qu'une forme de ponctuation dans ton vocabulaire et ne signifie pas nécessairement que le sujet de TD est une trivialité (sauf si?). Il parait qu'être doctorant c'est le "grade le plus élevé sur terre" et qu'à la riguur au dessus d'eux" [il y a] les druides"[2]. Mais un doctorant est avant tout un être par nature éphémère qui arpente pour une durée plus ou moins déterminée les open-spaces du laboratoire, il me faut donc remercié ceux qui sont partis

[2] https://www.youtube.com/watch?v=krzWc86Ub6I

avant moi : Adrien, Alexandre (J'espère avoir relevé dignement la difficile tâche de faire les TD d'Informatique théorique après toi.), Erwan (Je n'aurais finalement pas réussi à faire aussi bien que toi dans la longueur du manuscrit ; peut-être que Pascale m'en a été reconnaissante ?), Jad, Yassine ; et ceux que je laisse en partant: El Mehdi, Henri, Lucas, Manuel, Mathieu, Pierre (Au cas où personne ne te l'aurait précisé, la première étape de ta thèse, ce n'est pas de comprendre les réseaux de Pétri, mais bien de te développer un décodeur Pascale.) et Théo B. Pour ces derniers, j'espère que mon rôle de responsable IKEA vous permettra de profiter pleinement du "bon" open-space. Il est vrai que, malgré l'incontestable supériorité de l'open-space SC liée à sa vue sur la forêt, sa proximité avec la machine à café et l'évidente bonne ambiance proposée par ses habitants, il me faut tout de même remercier les occupants de l'open-space SB: Aaron, Antonin (Je tiens à préciser que, contrairement aux insinuations de certains qui se reconnaitront, Guacamole Vaudou est un excellent photoroman.), Elvire, Enzo, Gauthier (même si Isaure ne voudra plus monter dans ta voiture), Gurvan, Hakim, Joe, Jun (courage, tu es presque au bout !), Laura, Léo, Mahmoud (ou Mahmoud.e ? - carton rouge, c'est ça?), Mihir, Othmane, Paul, Sylvain, Théo E. et Yoann (à quand la prochaine partie de babyfoot ?). À Guillaume, pour nous rappeler que le CROUS, parfois y en a marre. À l'affable mafia grecque, Maria, Stergios et tout particulièrement Stefania pour ta générosité. À Brice, maintenant doublement exilé au deuxième étage dans un bureau plus grand que mon appartement et accessoirement quelque part au Canada, pour ton accueil, tes parties de jeux, et les discussions en balade dans la forêt de Massy.

Bien sûr, même si la frontière entre collègue et ami est parfois inexistante, j'ai aussi pu compter, lors de ces trois années, sur de nombreuses personnes qui ne sauraient porter le qualificatif de collègue. Parmi ces derniers, il en est qui ont aussi eu cette idée saugrenue de réaliser une thèse : Elena, Martin, Pablo, Valentin, nous avons chacun vécu une thèse très différente, mais je crois qu'il n'y a personne avec qui j'ai pu discuter plus librement de ma vie en thèse qu'avec vous. Pour autant, une thèse ne se résume pas au contenu de ses pages, ni aux discussions qu'elle a pu engendrer. Une thèse, c'est aussi tout le reste, tout ce qui n'est pas la thèse. Je tiens à remercier tous ceux qui m'auront permis d'oublier par moment la tâche que représente une thèse. À tous les membres du Centrale PARC, parce que comme l'a si admirablement dit Jean-Pierre Rives, "le rugby, c'est l'histoire d'un ballon avec des copains autour et quand il n'y a plus de ballon, il reste les copains." Au groupe Rando pour avoir réussi à me faire sortir du lit de bon matin des dimanches de pluie. Au groupe de la tour d'argent, Charles, Elena, Lucas, Martin, Van, en espérant ne pas trop rater de 21 pour continuer de partager cette histoire commencée par ce chapitre étrange de la classe préparatoire. Aux amis d'ici et d'ailleurs : à Adam, dommage que télémagouille ne soit qu'un sketch ; à Agathe, pour ces restaurants partagés avec ton cher Titi ; à Alexandra, princesse dans son château ; à Bastien le numismate, pour ton amitié depuis que j'ai des souvenirs ; à Clara, parce qu'on peut se perdre de vue pour mieux se retrouver ; à Gata, l'éternelle étudiante ; à Guibre, j'ai encore les règles de la coinche notées pour la prochaine partie ; à Loulou, on n'avait peut-être pas l'air des couteaux les plus aiguisés du tiroir, mais on les aura tous fait médire ; à Mathilde, il faudra quand même que je vienne voir chez toi, même si les vendredis au bar, c'est une bonne idée aussi ; à Nico, parce que c'est vachement mieux Montrouge que le Japon ; à Océfou, reste fidèle à toi-même, tu éclaires ton monde ; à Rem, oui, je sais qu'être doctorant, c'est une forme d'esclavage volontaire ; à Pablo, il fut un temps, il suffisait de traverser le couloir, maintenant, il s'agit de traverser la rue, mais je suis certain que notre complicité continuera même s'il faut traverser le monde ; to Shreya,

I hope you will come to Paris while its main attraction is still there ; à Simon, après la F422 où tu dormais sur ta planche de snow, c'est tout de même plus agréable de venir te voir avec Van aux portes de l'Île-de-France pour jouer à lancer des haches, j'ai d'ailleurs toujours les clefs de chez vous ! à Valentin, le romantique au grand cœur, tu croyais en moi, je croyais en toi, résultat, on sera chacun allé au bout d'une thèse ; à Yanis, une mais pas quinze ?

Merci à toutes celles et à tous ceux qui ont fait le déplacement jusque Gif-sur-Yvette pour venir m'écouter présenter mes travaux cette après-midi du 22 novembre, je crois qu'il n'y a rien de plus incroyable que de voir une salle d'amphithéâtre remplie pour sa présentation de thèse.

En ouvrant un manuscrit de thèse, les remerciements sont (trop) souvent la seule partie pleinement compréhensible (et donc que l'on prenne le temps de lire). Ainsi, si jamais vous ouvrez le manuscript d'Alexandre,[3], vous y découvrirez qu'il est possible de remercier un compositeur disparu depuis plus de 250 ans, et dans celle de Brice [4], que l'on peut remercier un meuble de salon. Moi, je tiens à remercier ma douche (et par la même occasion, à m'excuser auprès de la planète), car c'est à moitié endormi sous l'eau chaude et prétendument en train de me laver que j'ai eu toutes les meilleures idées qui peuplent ce document. Isaure, j'espère que tu comprends enfin mon incompréhension devant tes douches éclairs.

J'ai écrit les pages de ce document entre juillet et septembre, autant dire que s'il est présentement lisible ce n'est pas réellement à moi que reviennent les mérites. Il me convient donc de remercier mes relecteurs, mes encadrants tout d'abord, mais surtout trois personnes externes à mon projet qui m'ont offert quelques heures de leur vie. À Marc pour avoir relu les constructions catégorielles. À Agathe pour avoir continué ta relecture après avoir découvert que l'image de Lego n'était qu'une décoration de début de chapitre et non mon sujet de thèse. À Isaure pour tes relectures minutieuses d'un document écrit dans un langage étrange et difficilement compréhensible (je ne parle pas de l'anglais pour les petits malins), il est vrai que relire ton mémoire sur la neige au Canada était une mission moins ardue. Si certains passages sont remarquablement bien écrits, c'est sûrement grâce à eux, mais si vous trouvez une faute de typographie ou une quelconque erreur, c'est fort probablement un passage qu'ils n'ont pas relu et j'en assume la responsabilité complète.

À mes cousines, mes cousins, mes tantes et mes oncles, oui à 26 ans, je suis toujours étudiant, si je gagne un salaire, non, c'est pas le luxe, oui, je vais en cours, mais non, c'est pas pour dormir au fond de la salle puisque c'est moi le guignol devant le tableau, oui, c'est normal que tu ne comprennes pas forcément tous les mots de mon titre de thèse, oui ça fait bientôt trois ans que je travaille sur ce sujet, non, je ne peux pas juste chercher la réponse sur internet.

À Mamie Françoise, merci pour ton soutien inébranlable et incommensurable (bien que papa dira sûrement qu'il suffit de compter les gâteaux au chocolat, mais je lui répondrai que ce n'est qu'une forme de jalousie), merci pour tous tes petits mots d'encouragement, quel bonheur que de rendre sa grand-mère si fière. À Martin, un frère parfois excentrique, parfois de mauvaise humeur, rêveur éternel incompris, merci pour toutes ces conneries qu'on aura faites en grandissant, mais qu'il faudra garder secrètes, après tout, nous voilà tous deux docteurs ! À Papa, pour ces discussions sur la plage, pour ces randonnées où l'on réalise que finalement, on n'est pas grand-chose

[3] Alexandre Goy. On the compositionality of monads via weak distributive laws. Logic in Computer Science [cs.LO]. Université Paris-Saclay, 2021. English. NNT : 2021UPAST080. tel-03426949

[4] Brice Hannebicque. Regularity of generalized stochastic processes. Probability [math.PR]. Université Paris-Saclay, 2021. English. NNT : 2021UPASM007. tel-03260689

devant l'immensité de la montagne, pour cette expression sur ton visage lorsque le jury a validé ma thèse, et pour tout le reste. À Maman, pour ton amour, pour ta gentillesse, pour ton dévouement avec trois gosses à la maison, pour ton admiration aussi, et, enfin parce que c'est vrai que je suis enfin un adulte. Merci à tous les quatre de m'avoir permis d'être qui je suis aujourd'hui, je vous aime.

Au cours de cette thèse, j'ai aussi appris à découvrir une autre famille. Merci à Joséphine, Philippe, Côme et Hippolyte pour votre accueil, pour ces repas et ces après-midi passées à jouer à Catan, ces discussions sur la pharmacie et ces vacances dans le Sud (et le moins Sud où on mange des frites).

J'en ai presque fini, mais il me reste une dernière personne à remercier : celle sans qui je ne me serais jamais levé le matin (et sans qui j'aurais aussi probablement oublié de me coucher de nombreux soirs), celle qui a su me forcer à prendre des vacances, celle qui comprend pourquoi je me lève parfois pour gribouiller des hiéroglyphes sur un bout de papier au milieu de la nuit, celle qui m'attend actuellement emmitouflée dans la couette pendant que je termine d'écrire ces lignes. Merci Isaure. Merci pour ton incroyable patience. Merci pour ta complicité. Merci pour ton amour. Merci d'être toi. C'est un chapitre de ma vie qui s'achève ici, mais j'ai hâte de découvrir ensemble tous ceux qu'il nous reste encore à écrire.

Contents

List of figures

Notations

We summarize here the notations used throughout the dissertation for two reasons. First, we fix notations of general notions that vary between authors. Secondly, this glossary can be used as a look-up table for notions specific to our topic and related to our contributions. Nonetheless, all notions with be defined through the dissertation.

General notations

$\mathbb{N}$	Set of integers
$i..j$	Subset of integers between i and j included (assuming $i \leq j$)
$\mathbb{R}$	Set of real numbers
$\subseteq$	Inclusion of sets
$\sqcup$	Disjoint union of sets
$\times$	Carthesian product, also graph product
$X/\sim$	Quotient of X by the equivalence relation $\sim$
$[\![-]\!]$	Multiset of elements
Σ	Alphabet
$\Omega = (S, F)$	Data signature (Definition 57)
$\mathscr{M}$	Multiset data type
$\mathscr{A}$	Algebra (Definition 58)
ρ	Profile of a function (Definition 57)
O	Big O notation (complexity)

Category theory

$\mathscr{A}, \mathscr{B}, \mathscr{C}$	Categories
$\mathscr{O}(\mathscr{A})$	Objects in the category $\mathscr{A}$
$\mathscr{A}(A, B)$	Set of morphisms from A to B in the category $\mathscr{A}$
$A \to B$, $B \leftarrow A$	A morphism from A to B
$\circ$	Composition of morphisms
1_A	Identity on the object A
A, **B**, **C**	Small categories
Set	Category of sets and functions
$\varnothing_{\mathscr{A}}$	Initial object in the category $\mathscr{A}$ (Definition 6)

$\mathbb{1}_{\mathscr{A}}$	Terminal object in the category $\mathscr{A}$ (Definition 6)
$!_X$	Unique morphisms from X to the terminal object $\mathbb{1}_{\mathscr{A}}$ in the category $\mathscr{A}$
$A \hookrightarrow B$, $B \hookleftarrow A$	Monomorphism from A to B (Definition 12)
$\mathscr{M}$	A class of monomorphisms
$A \rightharpoonup B$	Partial monomorphism from A to B

Graphs and graph rewriting

Graph	Category of graph and graph morphisms
Σ-**Graph**	Category of graph arc-labeled on Σ (Notation 5)
TG	Generic type graph
$\mathbf{Graph}_{\mathrm{TG}}$	Category of graphs typed over TG (Definition 19)
Ω-**AGraph**	Category of attributed graphs on the signature Ω (Definition 63)
ATG	Generic attributed type graph (Definition 64)
Ω-$\mathbf{AGraph}_{\mathrm{ATG}}$	Category of attributed graphs typed over ATG (Definition 64)
$u \rightsquigarrow v$	A path from u to v
$u \overset{w}{\rightsquigarrow} v$	A path from u to v labeled by w
$G \Rightarrow^{r,m} H$	Direct derivation (Definition 25)
$\mathfrak{R}$	Relabeling set
$r = L \rightharpoonup R$	Generic notation for a rule (Definition 23)

Combinatorial maps

$\mathbb{D}$	Set of dimensions (subset of $\mathbb{N}$)
0,1,2,3	Dimensions zero, one, two, and three
κ, $\kappa(L,R)$	Free symbol for the joint representation, joint representation (Definition 36)
X_{+2}	Set of pairs (i,j) in $X \times X$ such that $i+2 \leq j$ (assuming $X \subseteq \mathbb{N}$)
$\mathbb{D}$-**Graph**	Topological graphs (Definition 28)
$\mathbb{D}$-**CGraph**	Combinatorial graphs (Definition 32)
$\mathtt{I}_G(i)$	Incident arcs constraint for the graph G and the dimension i (Definition 29)
$\mathtt{w\text{-}I}_r(i)$	Weak incident arcs condition for the rule r and the dimension i (Definition 33)
$\mathtt{I}_r(i)$	Incident arcs condition for the rule r and the dimension i (Definition 34)
$\mathtt{O}_G(i)$	Non orientation constraint for the graph G and the dimension i (Definition 30)
$\mathtt{O}_r(i)$	Non orientation condition for the rule r and the dimension i (Definition 37)
O, **N**	Set of oriented, non-oriented dimensions
$\mathtt{C}_G(i)$	Cycle constraint for the graph G and the dimension i (Definition 31)
$\mathtt{C}_r(i)$	Cycle condition for the rule r and the dimension i (Definition 40)
E	Set of exchangeable dimensions
$\mathbb{W}$	Finite set of words on $\mathbb{D}$
$\mathbb{W}$-**Graph**	Pattern graphs (Definition 42)
$\mathbb{E}_\Sigma$	Embedding functor (Definition 41) on Σ
$\mathbb{E}_\mathbb{D}$	Embedding functor on words

π_Σ	Projecting functor (Definition 41) on Σ
$\pi_{\mathbb{D}}$	Projecting functor on dimensions
$\mathbb{P}_{(\mathbb{D},\mathbb{W})}$	Pattern functor (Definition 48)
$\mathscr{S}$	Generic notation for a rule scheme (Definition 46)
$\iota(-,P)$	Instantiation functor
$\iota(\Pi,P)$	Graph scheme instantiation
$\iota(\mathscr{S},P)$	Rule scheme instantiation (Definition 47)
$\langle o\rangle$	Orbit type (Definition 56)
$\langle o\rangle(v)$, $G\langle o\rangle(v)$	Orbit of the node v (in the graph G) (Definition 56)
π	Embedding functions, target of the attribution arc encoding an embedding
Π	Familly of embeddings
$\mathscr{T}$	Topological core of an embedded graph (Definition 72)
ETG	Generic embedded type graph (Definition 69)
$E^{\leqslant}_G(\pi)$	Partial embedding constraint for the graph G and the embedding π (Definition 71)
$E_G(\pi)$	Embedding constraint for the graph G and the embedding π (Definition 71)
$\mathscr{G}$	Generic Jerboa graph scheme (Definition 87)
$\iota^{\langle o\rangle}(\mathscr{G},O)$	Graph scheme instantiation in Jerboa's sense (Definitions 88 and 89)
$\iota^{\langle o\rangle}(\mathscr{S},O)$	Rule scheme instantiation in Jerboa's sense (Definitions 88 and 89)
Poi	Sets of points of interest (Section 8.2.3)

Introduction

On the importance of inference

The term inference relates to the establishment of a conclusion from facts or premises. One could even say that inferring is no more than making an educated guess. Inference has been a subject of philosophy and logic, formalizing human reasoning. Inference is traditionally split into deductive reasoning, which obtains the conclusion through logic such that the conclusion holds only if the premises do, and inductive reasoning, which obtains the conclusion from observations without guarantees of correctness on the conclusion. Besides, inference characterizes both the obtained conclusion and the process of reaching this conclusion. In psychology and linguistics, inference describes a cognitive mechanism related to the interpretation of facts, adding information that is not given and that may not even be implied. In this sense, if the police knock at your door, you may infer that you have an issue with the law.

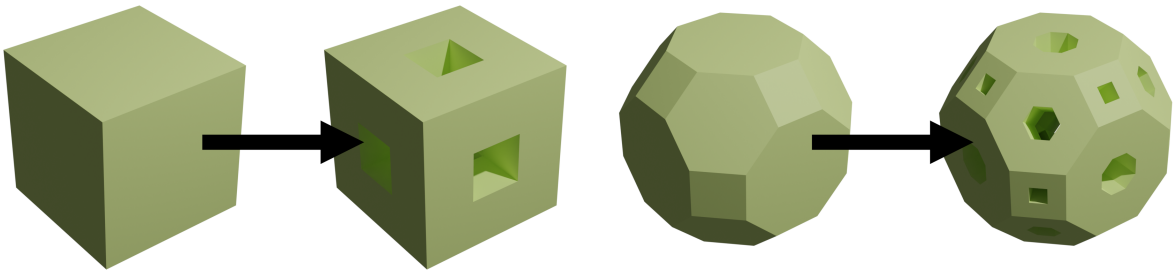
Within computer science, the term inference relates to expert systems used in artificial intelligence or statistical inference, which is at the core of machine learning. The former deduces knowledge based on already existing knowledge, while the latter tries to retrieve information on a population from a sample. In that sense, inferring means retrieving some truth or information hidden in the data.

Inference in geometric modeling

Geometric modeling deals with the representation and manipulation of geometric objects. This sub-field of computer science has already studied the topic of inference. For instance, machine learning can be used to deduce the parameters of rules in inverse procedural modeling [170] or the geometric values used in a subdivision scheme [123]. In the former case, the idea is to obtain a procedural generation of objects to alleviate the task of obtaining complex objects and scenes with several layers of redundancies. In the latter case, the idea is to reconstruct a refined object from a coarse mesh. These two approaches share the idea that a symbolic description of the transformation is given and that numerical values are the subject of the inference.

In a broader approach, inference can be used as a tool to alleviate the cumbersome task of implementing dedicated algorithms. Instead of simply inferring values, a more general solution provides a set of possible modifications and deduces the sequence of transformations that yields an object. This approach has been used in constructive solid geometry [162, 109] or CSG. In CSG, an object is obtained from geometric primitives that describe simple shapes (e.g., cubes, spheres, or prisms), a finite set of boolean operations (i.e., union, intersection, and difference), and geometric transformations (e.g., translations, rotations, or scaling). Therefore, an object is entirely described

by a tree whose leaves are the geometric primitives and whose nodes are the operations. Inverse CSG is the process of inferring a tree that describes an object. In inverse CSG, the set of possible operations is fixed, which is well suited to reconstruct a target object starting from nothing but does not provide an adequate solution to obtain a modeling operation, i.e., the transformation from one object to another. Such approaches solve the problem of providing a construction sequence for a given object.



(a) (b)

Figure 1: Inferring a geometric modeling operation: (a) before and after objects used for inferring the operation, and (b) application of the inferred operation.

In this thesis, we strive to establish a solution for users who are not experts in geometric modeling to design operations and software for their dedicated field of expertise. For instance, we want to infer how to modify the cube given on the left-hand side of Figure 1a to obtain the object on the right-hand side. The essential element in Figure 1a is the arrow between the two objects, i.e., the modeling operation. Indeed, the inferred result should provide a solution to modify other objects, such as the volume on the left-hand side of Figure 1b.

Regardless of the goal and methods involved, all these inference mechanisms inherently exploit a predefined formalism. We propose to use the formalism of graph transformation rules which extends the rewriting of words to non-linear structures, providing a more versatile and expressive framework than L-systems and empowering formal reasoning through several abstraction layers.

Graph transformations

In this thesis, we infer modeling operations within the formalism of Jerboa's rule-based language. Jerboa [12] is a platform for prototyping dedicated geometric modelers, i.e., software for manipulating geometric objects. The platform has been successively used for modelers across various application domains, such as architecture [30], simulation of plant growth [21], and soil analysis [74]. The key component of Jerboa that we exploit in this dissertation is the rule language that allows for the design of modeling operations.

In abstract rewriting systems, a rule is intuitively written $L \rightarrow R$ and describes the transformation of L into R. The idea behind rules is to factorize the transformation. In other words, L is reduced to the smallest part that should occur to enable its replacement by R. In that sense, the transformation now occurs within a bigger context G. Then, the rule rewrites G into H by finding an occurrence of L and replacing it with R. Loosely, we can describe H as $G - L + R$. We obtain the intuitive description of Figure 2.

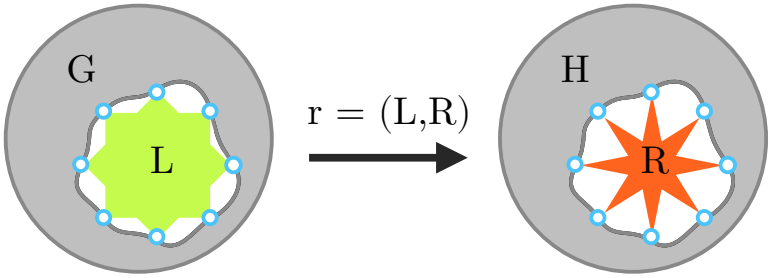


Figure 2: Intuitive description of a rewriting step.

In this intuitive description, we depicted some elements in blue. These elements, identical in G and H, belong to the rule, i.e., to L and R, and describe how to reconnect the modified part within the more general context. When rewriting strings (i.e., sequences of letters), applying a rule is achieved by first skimming through the sentence until the sequence of letters constituting L is found, then removing it, and finally replacing it with the sequence of letters encoded by R. Two blue elements are needed for string rewriting, one at the beginning of the replaced string and one at its end. In the formalism of graph transformations, the elements L, R, G, and H are graphs, i.e., a set of nodes linked via a set of arcs. In such a context, our intuitive description raises several questions, namely:

- How do we find an occurrence of L in G?
- How do we remove L from G?
- How do we replace L by R?

These questions have been answered through the construction of algebraic approaches to graph transformations [55, 57, 99]. The needed notions are related to category theory, and we will present the answers to these questions in Chapter 2. Note that this dissertation's purpose is not to pursue fundamental results in the theory of graph rewriting but rather to specialize graph transformations to a specific class of graphs used in geometric modeling. Our approach is similar to that of Jakob Andersen and Daniel Merkle but for geometric modeling instead chemistry. Indeed, we are "just trying to bend the formalisms a bit [...] and figuring something out in the middle."[5]

The question of inference is no stranger to the graph rewriting community, as attested by the panel discussion at the 2022 GCM workshop, "Learning Graph Transformation Rules" [97] and the keynote presentation by Elizabeth Dinella. In her talk, Dinella presented Hoppity [47], a bug detection and repair tool that exploits graph transformations as a learning mechanism. In this tool, the inference relates to retrieving the sequence of transformations to apply, while our ambition is retrieving the rules themselves. To reverse engineer software systems, approaches that retrieve rules rather than assuming a given set of rules and retrieving transformation sequences have been used [99, Chap. 8]. In a dynamic approach, test runs provide an understanding of the possible behavior of the system by retrieving visual contracts [1]. These visual contracts are typed attributed graph transformation rules, meaning they describe both structural modifications, i.e., graph structure, and modifications of the attribute values. The inferred rules are then generalized

[5] Jakob. L. Andersen. "Chemical Graph Transformation and Applications". December 4, 2020. GReTA Seminar #2. https://www.irif.fr/~greta/event/dec4th2020-andersenmerkle/.

by removing all elements not needed to perform the modification and by using multi-patterns to obtain more abstract rules. Our approach has essentially the same ambition but analyzes generalized maps instead of traces of Java programs, Jerboa's domain-specific language instead of UML diagrams, and abstracts the transformations via schemes instead of multi-patterns.

Topology-based geometric modeling

Our inference mechanism retrieves modeling operations from generalized maps. A generalized map is a combinatorial structure defined as involutions on a set of elements called darts [119, 120, 43]. This model belongs to the domain of topological-based geometric modeling.

In topology-based geometric modeling, we are interested in both the outer and the complete internal structure of objects. Indeed, inner parts play a key role in applications where volumetric properties are essential: geology [74], architectural buildings [102], or physical simulation [186, 17]. The representation of the inner parts relies on a subdivision of the object into topological cells: vertices correspond to cells of dimension 0 (see Figure 3b); edges link together vertices and form cells of dimension 1 (see Figure 3c); faces are the cells of dimension 2, bordered by edges (see Figure 3d); volumes are bounded by faces and represent the cells of dimension 3 (see Figure 3e); in higher dimensions, the cells of dimension i are enclosed by cells of dimension $i-1$.

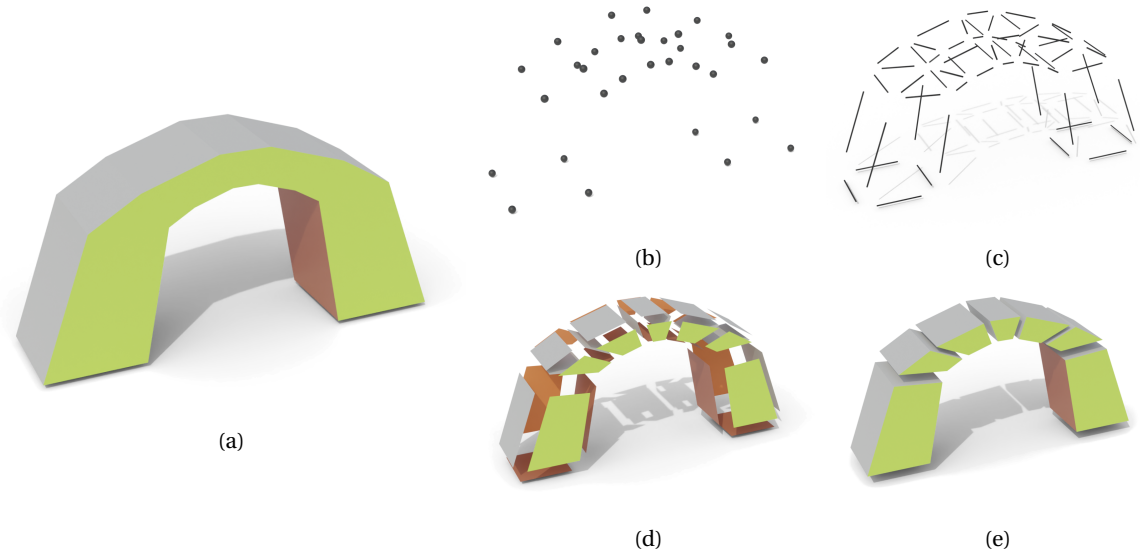


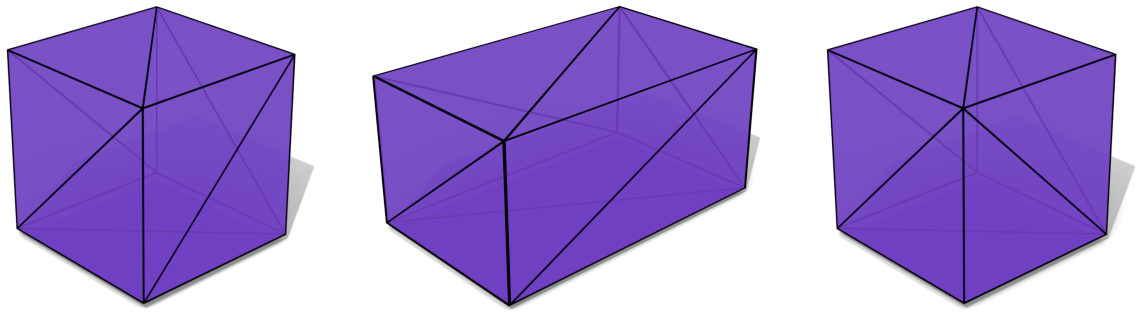
Figure 3: Cells in a 3D object: (a) a 3D object, (b) its vertices, (c) its edges, (d) its faces, and (e) its volumes.

The reconstruction of an object is not the only cumbersome task in geometric modeling. Designing *modeling operations* can also present a challenge, requiring substantial implementation and debugging efforts (either in a programming language or in a higher-level tool). Borrowing results from rewriting systems, L-systems allow the representation of geometric objects as words, such that a modeling operation can be described as rewriting the word. In this context, the inference of a modeling operation corresponds to the inference of a rewriting rule. In [169], the authors present the definition of rules as the "key challenge of procedural modeling" and exploit machine learning to retrieve both the word representing an image and the L-system that can derive this word. However, not all objects admit a simple representation as words, hindering the possibility of effectively retrieving a modeling operation if the object cannot be properly encoded.

We are mainly interested in the topological relations between these cells. These relations are the *adjacency relations* and the *incidence relations*. A cell of dimension i (or i-cell) is incident to a cell of dimension $i+1$ if it belongs to its boundary. In this case, the $i+1$-cell is also said to be incident to the i-cell. For instance, an edge is incident to a vertex if the vertex is one of its endpoints, while a face is incident to an edge if the edge belongs to its boundary. Two i-cells are adjacent if they share a common cell in their boundary, i.e., if they are incident to the same cell. For example, two edges are adjacent if they share a vertex or belong to the same face.

Among the various combinatorial models, Jerboa exploits the model of generalized maps [43]. The specificity of our approach is that we consider generalized maps as graphs, describing the darts as nodes and the involutions as arcs labeled with dimensions. This graph-based approach allows the design of geometric modeling operations as rules in the framework of graph transformations.

The graph's structure encodes the topological structure of the object. This topological content is completed with geometric information to represent the object. Minimally, we need to provide positions to the topological vertices such that the object can be displayed. Other values may be required based on the application domain such as colors, textures, normals, physical properties, or semantical information. All this non-topological information is encapsulated via embedding values added to the topological cells of the object. In particular, we may subdivide a cell into smaller cells to describe a variation in embedding values. The distinction between topology and geometry is illustrated in Figure 4. The object in Figure 4a is a cube where each square face has been split into two triangles. The object in Figure 4b has the same topology as the cube in Figure 4a but has been stretched, resulting in a different geometry. The object's vertices in Figure 4c have the same position as the vertices from Figure 4a, but an edge has been flipped (on the right-hand side of the cube), meaning that the two objects have a different topology.



(a) (b) (c)

Figure 4: Difference between topology and geometry: (a) an object, (b) an object with the same topology as (a) but with a different geometry, and (c) an object with the same geometry as (a) but with a different topology.

Contributions

On the use of graph transformations for geometric modeling

Jerboa's rule language offers a specialization of graph transformations for topology-based, and more generally, geometric modeling with the ambition of providing a tool for generating modelers. This statement raises questions related to the expressiveness and properties of the language.

▶ *What assets of graph transformation can help formalize geometric modeling operations?*

Rewriting, in general, and graph rewriting, in particular, allows formal reasoning about transformations. Stricto sensu, graph transformations operate directly on the graph structure, whereas geometric modeling operations usually have a higher level of generality. For instance, a modeling operation describes the subdivision of a surface without accounting for the actual surface representation. In comparison, a rewriting rule relies on finding an exact structure within the modified object. We will show how to bend the formalism of graph rewriting to obtain a framework in which more general operations can be written. More precisely, we propose a generalization via product-based functors that mimic a global relabeling for abstracting the topology and a rule completion mechanism with two-layered terms in attributed rules for the geometry.

▶ *Can we use graph rewriting as a technique to represent modeling operations on models other than generalized maps?*

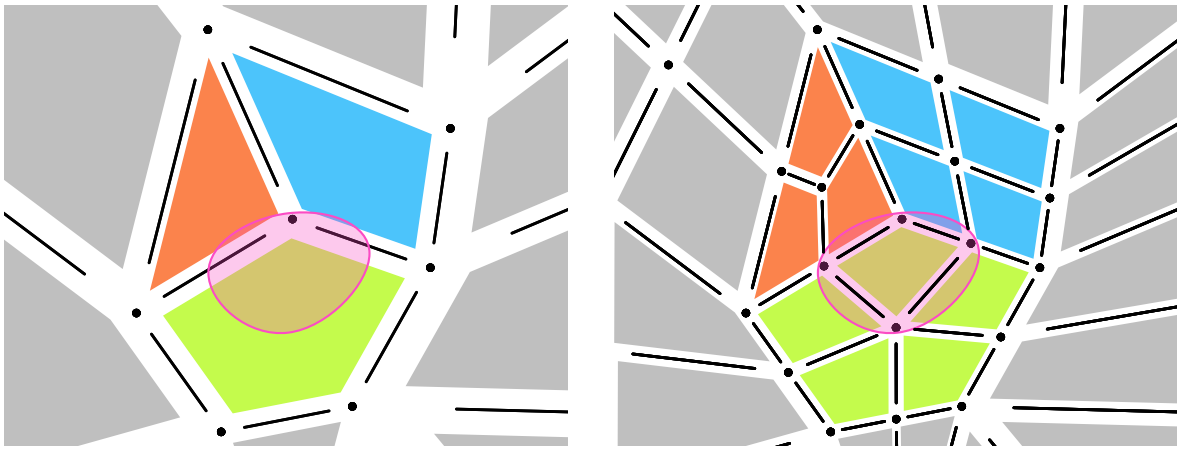
Generalized maps are one of the most regular models within topology-based geometric modeling since all permutations are involutions. Compared to generalized maps, the advantage of oriented maps is a compact representation of objects (using twice as less darts) that is to the detriment of regularity in dimension (one dimension has a particular treatment), making reasoning more difficult. Oriented maps are supported by efficient implementations such as CGAL [42] or the CGoGN library [113] and are usually preferred for industrial applications. We will demonstrate that a unified framework encoding both generalized and oriented maps is possible, such that modeling operations on oriented maps can also be designed as graph rewriting rules.

▶ *What kind of guarantees can be obtained from working with graph transformations?*

Combinatorial maps constitute a mathematically sound description of objects. Thus, modifications of a well-formed object should produce an equally well-formed object. We will present conditions on (extended) rules that ensure the preservation of the consistency constraints derived from the reformulation of the models as graphs. In particular, we will ensure consistency at the rule level to provide feedback when designing rules.

On the inference of modeling operations

We stated that the inference process was highly dependent on the formalism of what we sought to retrieve, to the point that different methods should be exploited for different formalisms. Within topology-based geometric modeling, we distinguish the topological and geometric content. The topology of the represented object is encoded by the graph's structure, while the geometry is stored via values added to the topological cells. These two parts are conceptually distinct, which motivates us to study the inference of the topological and geometric modification separately and with different methods.



(a) (b)

Figure 5: Intuition for the inference of an operation: (a) a surface before the modification, and (b) the surface after the modification.

Considering the transformation suggested by Figure 5, we can infer a quad subdivision operation. Our brain looks for patterns, trying to give sense to the transformation. For instance, we can isolate the part highlighted in pink. This part corresponds to a face corner in Figure 5a, and a quad in Figure 5b. This intuition that each face corner yields a quad is verified everywhere on the surface, meaning that the operation most likely acts on an entire surface. We can also remark that each new quad inherits the color of the face corner, while the added vertices appear to be placed in the middle of the initial edges or faces. In a sense, we propose to simulate this human reasoning, which raises several questions.

▶ *How can we find the relevant topological regularities within the object and track their modifications through the operation?*

Any small pattern within the object may describe a topological regularity in the object. For instance, a pattern encoding an edge with two endpoints will happen for every edge within the object. The difficulty is to find the relevant pattern in the object that effectively describes the operation. We will see that such patterns can be found by folding the object along its topological relations. In particular, if a coherent folding of the instance before and after the modification can be found, the obtained pattern entirely encodes the operation, described as a graph transformation rule.

▶ *How can we retrieve an expression encoding the computation of the new geometric values?*

For the context of this work, we assume to know the embedding values associated with each cell before and after the operation (even if the operation modifies the cells). We are searching for the formulas that yield the values in the after instance based on the before instance. The difficulty here is to conciliate the geometric formula with the topological modification to obtain a graph transformation rule. We will explain how to solve this issue by introducing hypotheses on the form of the computation, namely linearity, and the values used for computation, introducing the notion of values of interest.

Organization of the manuscript

This dissertation is divided into two main parts: Part I about the formal language and Part II about the inference of geometric modeling operations. Since the first part of the thesis will mainly focus on theoretical aspects of exploiting graph transformations to handle geometric modeling operations, we will start the presentation with a chapter about geometric modeling. Globally, the manuscript is organized as follows.

- In Chapter 1, we discuss the models used to represent objects in geometric modeling with an emphasis on combinatorial models used in topology-based geometric modeling. We also discuss how to enforce soundness in the design of geometric modeling operations, mostly in higher-level solutions relying on a formal description of modeling operations.

Part I provides a formalization of geometric modeling operations as graph transformation rules and consists of the following chapters.

- We start the formal part of the dissertation with preliminaries on category theory and graph rewriting. Chapter 2 presents the notions from category theory needed through the first part of the dissertation and concretizes them in the category of graphs. We also discuss the double-pushout approach to graph rewriting within adhesive categories and some applications of graph rewriting.

- Chapter 3 defines generalized and oriented maps as a particular subclass of labeled graphs subject to constraints and provides conditions on rules to preserve the consistency of the underlying model. This chapter corresponds to the first part of [139].

- In Chapter 4, we extend the framework of Chapter 3 with rule schemes that allow for topological abstractions of the rules, hence obtaining transformations representing modeling operations adequately. The extensions rely on a product of graphs encoding a relabeling function to be applied globally on the modified part of the graph. We also lift the consistency conditions from rules to rule schemes. This chapter corresponds to the second part of [139].

- For the model of generalized maps, we study the addition of embeddings via node attribution in Chapter 5. Since embeddings are functions from topological cells to an embedded space, we add constraints that an attributed graph should satisfy to be an embedded generalized map. From these constraints, we derive conditions on rules which, in turn, prove to be partially counter-intuitive from a user perspective. Therefore, we use an atypical adaptation of the rewriting mechanism to automatically complete rules based on the modified part of the object, obtaining a suitable description of modeling operations. The results of this chapter have been published in [3].

Part II presents the inference of geometric modeling operations with Jerboa and consists of the following chapters.

- In Chapter 6, we present the components of the Jerboa platform and redefine some notions from Part I from a practical perspective. The main purpose of this chapter is to provide the

necessary tools to understand our method for the inference of operations presented in the two following chapters. In particular, we present the instantiation of rule schemes in general terms, similar to our description in [138].

- Chapter 7 describes the topological folding algorithm used to infer the topological part of an operation from an example. We present theoretical and practical validations of the method and discuss some unforeseen advantages. The chapter corresponds to an extension of [138], enabling the inference of local modifications.
- Finally, Chapter 8 augments the inference mechanism with the deduction of embedding expressions formulated as a constraint satisfaction problem.

Reading guide for the manuscript

As hinted by the organization of the manuscript, the first part of the dissertation is mainly about the formalization of geometric modeling operations with graph transformations, while the second part discusses the inference of modeling operations, presenting topics closer to implementation with an emphasis on computer graphics and geometric modeling. These two parts were written as standalone discussions. Thus, a complete understanding of Part I is not a prerequisite to Part II. At the same time, the first part can also be considered purely as a contribution to the construction of a formal framework for topology-based geometric modeling. Nonetheless, the reader will find that some of the choices in the formal part are driven by practical concerns, while some intuitions for our inference mechanism come naturally from the concepts related to graph transformations.

Each chapter starts with a personal note giving insights into the motivations, context about how the results of the chapter were established, or advice on how to read the chapter.

Chapter 1

Rule-based operations in topology-based geometric modeling

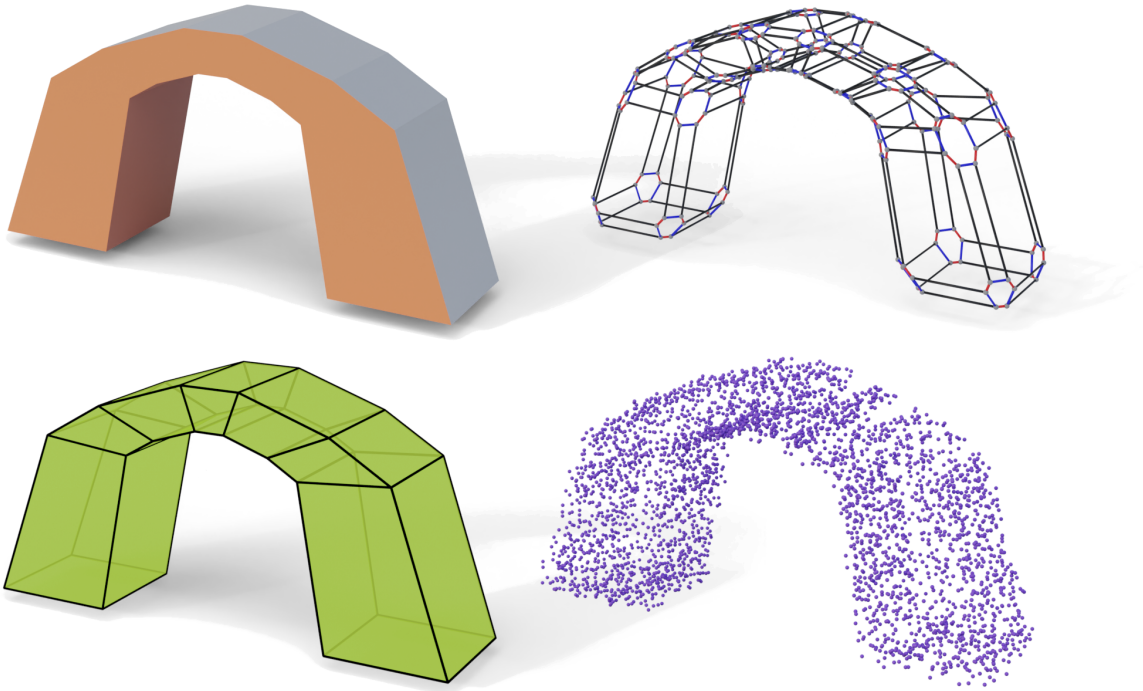


Figure 1.1: Different representations of the same object: (from left to right, from top to bottom) an object, a generalized map, a mesh, and a point cloud.

Personal note on the chapter

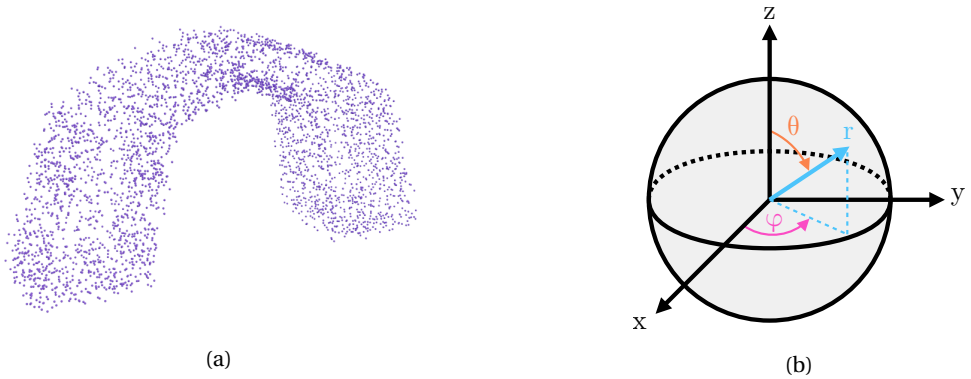
My dissertation is essentially divided into two parts. The first one presents the foundations of our dedicated language within the framework of graph transformations. The second one deals with the inference of modeling operations within this dedicated language. Thus, we work with a somewhat heterogeneous toolbox, with graph-transformation nails and hammers mixed in with geometric-modeling screws and screwdrivers. However, our geometric-modeling tools will only be used extensively later in the presentation, meaning that our formal ones will mostly leave the toolbox at the beginning. In order to provide context to a reader less accustomed to geometric modeling, we briefly present the domain in this chapter. While presenting the bigger picture, we will also delimitate the scope of this thesis, namely rule-based operations in topology-based geometric modeling.

Contents

Computer graphics correspond to the field of computer science that studies methods used to display images on a device, e.g., a computer screen. These methods range from solutions for representing and editing objects, animating them, generating an image from a scene, handling light interaction, or image editing. In this dissertation, we focus on the representation and edition of objects, i.e., geometric modeling. More precisely, *geometric modeling* corresponds to the mathematical representation of n-dimensional objects (virtual or real) in a discrete setting; it encompasses computer-aided design and manufacturing with applications across mechanical engineering, animated movies, video games, geology, and architecture.

1.1 Object representations in geometric modeling

Several approaches exist to represent *geometric objects*, with different distinct advantages justifying their uses in specific applications. For instance, triangle soup is a commonly used model for rendering virtual views because graphic processors are optimized to render triangular primitives [127, Chapter 12]. Animation deals with images changing over time, e.g., the motion of objects and characters in an animated movie or a video game. Articulated objects are represented using a hierarchical structure of the object's parts (called skeleton or rig) and a surface (typically a mesh) associated with each bone of the skeleton. To animate an articulated object, one can animate the skeleton and render the surface mesh on a set of poses [6].



(a) (b)

Figure 1.2: Structures for representing geometric objects: (a) a point cloud representation of an arch, and (b) a parametric representation of a sphere.

1.1.1 Point clouds

In this thesis, we are interested in representations used for designing and editing objects. For example, point clouds represent objects as a sample of points in the space that belongs to the object, typically on the external surface (see Figure 1.2a). *Point clouds* are easy to acquire via laser-telemetry or photogrammetry. Point clouds only consist of a set of points with data attached to the points. No other information is stored, which can prove efficient memory-wise. In particular, point clouds do not handle any relation between the points, which eases the processing of the cloud in machine learning approaches [84], but prevents their use for other applications. In such a case, the set of points is converted to a surface via surface reconstruction algorithms [18]. For instance, we can try to approximate a surface by providing a mathematical equation.

1.1.2 Equationnal representation of a surface

An equation can either provide an implicit or parametric description of the surface. The *implicit surface* describes an object as the zero-set of a three-dimensional function. For instance, the equation zero-set of the function $F(x,y,z) = x^2 + y^2 + z^2 - r^2$ describes a sphere or radius r. A *parametric surface* describes the object via a function $\mathbb{R}^2 \rightarrow \mathbb{R}^3$. The parametric description of a sphere is $f(\theta,\phi) = (r\sin(\theta)\cos(\phi), r\sin(\theta)\sin(\phi), r\cos(\theta))$ with $(\theta,\phi) \in [0,\pi] \times [0,2\pi]$ (see Figure 1.2b). If the equation of a sphere is relatively simple, retrieving the exact equation of more complex surfaces might prove challenging. A solution might be to interpolate the points with spline functions to obtain a non-uniform rational basis spline, or NURBS, that describes the object from a finite set of control points [67]. NURBS inherently describe smooth shapes but are not necessarily well handled by graphical hardware, which tends to favor meshes.

1.1.3 Meshes

A mesh represent an object as a set of vertices, edges, and faces. The vertices are positions in space (which can also store additional information, such as color or normal vectors). The edges are line segments between vertices, and faces are a closed set of edges. Several representations can be used to store the mesh. The *vertex-vertex* model [165] only stores vertices, relying on graph rotation systems [53] to associate an oriented circular list of neighbors to its vertices. The prevalent *face-vertex* model stores a collection of vertices and a collection of faces, where each face stores its set of vertices. Note that a mesh is a geometric description of an object, and the topological information is not directly present (if even present). Recall from the introduction that the topology of a mesh describes the adjacency and incidence relations between the faces, edges, and vertices of the mesh, without accounting for the position of the vertices.

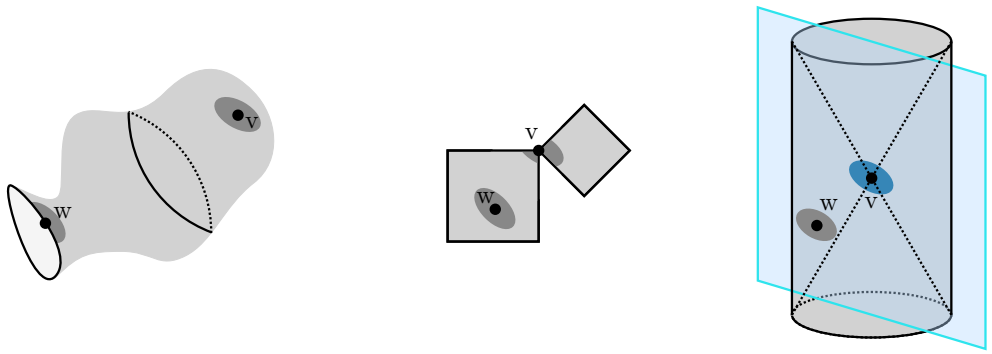
1.2 Topological description of an object

Topology studies shapes or, more precisely, invariants in continuous deformations of spaces. In this dissertation, we view topology via a combinatorial description, i.e., we study relations between the object subparts. The object is subdivided into topological cells of various dimensions. A cell of dimension i, or i-cell, is essentially a ball of dimension i. For instance, a 0-cell is a vertex, a 1-cell is an edge, and a 2-cell is a face. The cell decomposition of an object provides a discrete description of the object, turning its topological and geometric properties into algebraic and combinatorial constructions. Such constructions belong to the field of algebraic topology, dealing with questions such as persistent homology or the study of fundamental groups [95]. Although algebraic topology is a rather theoretic field of mathematics, it has (sometimes indirect) applications in geometric modeling and, more generally, computer graphics. See, for instance, the Topological ToolKit [176]. Similarly, abstract simplicial complexes constitute the beginning of the course by Keenan Crane at SIGGRAPH about discrete geometry [39].

1.2.1 Edge-based data structures

The subfield of geometric modeling which explicitly distinguishes the topology and the geometry is called topology-based geometric modeling. In topology-based geometric modeling, we focus on

the topological relations between subparts of the represented object. In 2D objects, these topological relations are usually encoded within the edge-based data structures [29]. First, Bruce Baumgart introduced the winged-edge structure [9, 10] to allow for a precise encoding of the topology. Quad-edges [82] and half-edges can be seen as refinements of winged edges, storing less information (meaning it should be recovered when needed). Half-edges are probably the most common of these structures, used, for instance, in OpenMesh [25] or Geometry Central [163]. These representations only allow describing manifold surfaces. A *manifold* is a locally Euclidean topological space, i.e., a space where each point has a neighborhood homeomorphic to the open ball (or half ball for the boundary points). For instance, the object of Figure 1.3a is a manifold, but the objects of Figures 1.3b and 1.3c are not manifolds. The half-edge structures can be twisted by chaining half-edges around non-manifold edges [163] bypassing the previous limitation and, thus, representing non-manifold surfaces. For instance, the radial edge structure [183] used in Blender [71] natively allows for non-manifold objects. Nonetheless, modelers build edge-based data structures when the surface connectivity needs to be modified. For instance, libigl [106] builds arrays similar to half-edges whenever remeshing-type operations are performed. The structure is then deleted, meaning it must be rebuilt for each remeshing operation. Note that all these structures can be seen as variations of the same idea, storing neighboring relations around (parts of) edges. The only difference is which relations are stored and which ones have to be recovered.



(a) (b) (c)

Figure 1.3: Manifolds and non-manifolds: (a) a 2D manifold with boundary, (b) a 2D object which is not a manifold nor a quasi-manifold, (b) a 3D object which is a quasi-manifold not a manifold.

1.2.2 Combinatorial maps

In [120], Pascal Lienhardt shows that such data structures are equivalent to the more formal n-dimensional maps. A complete conversion from half-edges to 2-dimensional maps has been given in [112]. In the literature, n-dimensional maps are also called combinatorial maps. In this dissertation, we call them oriented maps[1] to stress their orientation property. We will use '*combinatorial maps*' as a generic term that encapsulates several models (such as oriented maps, generalized maps, hypermaps, chains of maps), following the terminology given in [43]. From an algebraic perspective, combinatorial maps are often considered a collection of permutations on a set of atomic elements. The study of combinatorial maps comes from combinatorics to represent cell decompositions of a surface [178]. These maps are also used to draw graphs on a given surface,

[1] Therefore, Definition 1 corresponds to Definition 26 'n-map' in [43].

exploiting the combinatorial properties of the underlying graph, e.g., Euler's formula. Topological maps have also been defined as graphs embedded in a surface [117]. We will use combinatorial maps as a data structure to encode geometric modeling objects. We give the combinatorial definition of oriented maps as a reference for their graph counterpart defined in Chapter 3. We recall that a permutation is a bijection from a set to itself and that an involution is a permutation equal to its inverse.

Definition 1 (Oriented maps (adapted from [43]))**.** *An n-dimensional* oriented map*, with $0 \leq n$, is an $n+1$-tuple $\mathrm{M} = (\mathrm{D}, \beta_1, \ldots, \beta_n)$ where:*

- D *is a finite set of darts;*
- β_1 *is a permutations on* D *whose inverse* β_1^{-1} *is written* β_0*;*
- $\forall i \in 2..n$, β_i *is an involution on* D*;*
- $\forall i \in 0..n-2$, $\forall j \in i+2..n$, $\beta_i \circ \beta_j$ *is an involution.*

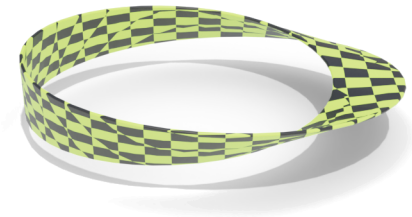


Figure 1.4: The Möbius strip is a surface that cannot be oriented.

In the algebraic definition of oriented maps, the permutations β_i encode the topological relations for the dimension i. Thus, the definition is not homogeneous in dimension since β_1 plays a distinct role compared to β_i for $i \geq 2$. Furthermore, an oriented map requires that the represented object is orientable, meaning that it cannot encode the Möbius strip (see Figure 1.4). A new model was defined to bypass this limitation [119]. This model, called generalized maps, handles the 0 dimension to bypass this limitation [119]. As a result, orientation is no longer required, and the number of darts is doubled.

Definition 2 (Generalized maps (adapted from [43]))**.** *An n-dimensional* generalized map*, with $0 \leq n$, is an $n+2$-tuple $\mathrm{M} = (\mathrm{D}, \alpha_0, \ldots, \alpha_n)$ where:*

- D *is a finite set of darts;*
- $\forall i \in 0..n$, α_i *is an involution on* D*;*
- $\forall i \in 0..n-2$, $\forall j \in i+2..n$, $\alpha_i \circ \alpha_j$ *is an involution.*

We draw the reader's attention to the meaning that must be given to a dart in a generalized map and an oriented map. Compared to a generalized map, the only definitional differences in an oriented map are the orientation of β_1 and the removal of dimension 0. Indeed, β_0 does not encode any topological meaning regarding the dimension 0. However, the dart's semantics changes the interpretation we should give to each model. Darts in a generalized map represent parts of topological vertices. In contrast, darts in an oriented map represent parts of topological edges (in 2D, the darts of an oriented map correspond to half-edges).

1.2.3 Representable objects

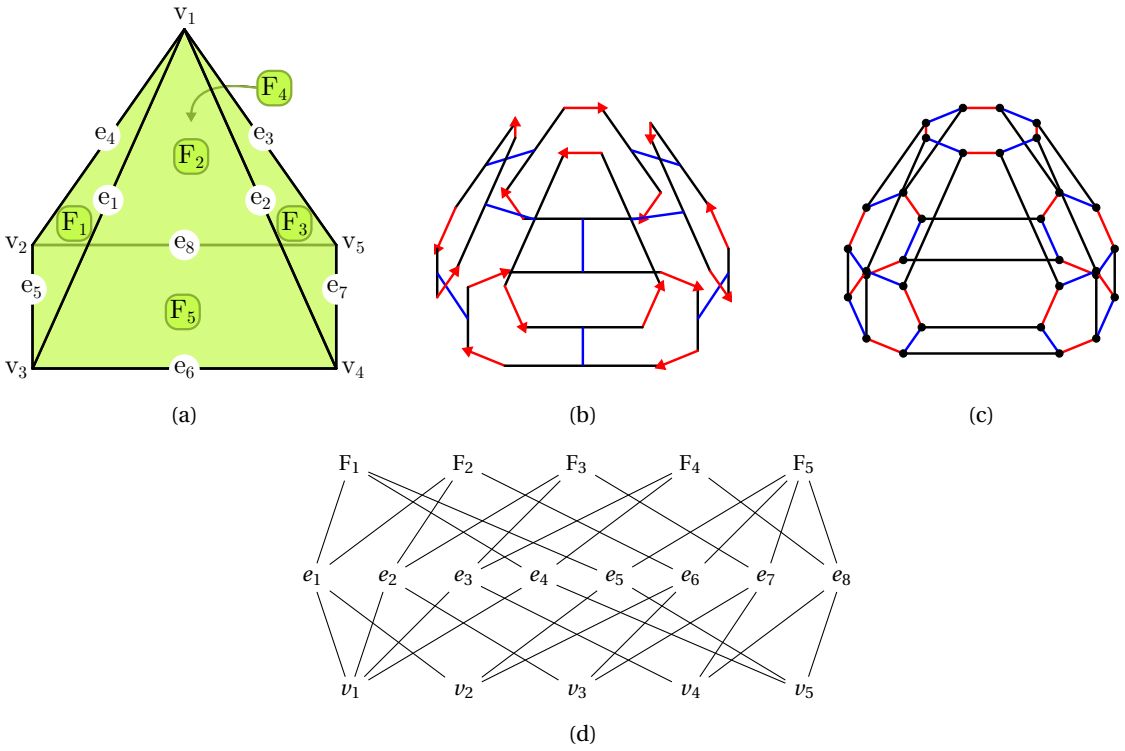


Figure 1.5: Representations of a pyramid: (a) a 2D object with identified cells, (b) representation as an oriented map, the darts are displayed as black lines, the permutation β_1 as red arrows, and the involution β_2 as blue lines, (c) representation as an oriented map, the darts are displayed as black dots, and the involutions α_0, α_1, α_2, as black, red and blue lines, (d) representation as an incidence graph, each cell is linked to its incident cells.

In both models, the topological cells are not directly present in the structure, as they would be in a representation with an incidence graph (see Figure 1.5). When working with arbitrary relations between the topological cell, the represented object belongs to the class of cellular complexes [95], which also encompasses non-manifolds. For oriented and generalized maps, the class of representable object corresponds to *quasi-manifold*, which can intuitively be described as cellular complexes where n-cells can only be glued along $(n-1)$-cells. In dimensions 1 and 2, manifolds and quasi-manifolds represent the same class of objects. Manifolds are strictly included in quasi-manifolds in dimensions greater than 3. As already discussed, the object of Figure 1.3a is a manifold and, thus, a quasi-manifold. The object of Figure 1.3b is not a quasi-manifold since the two faces are glued along a vertex. The 3D object of Figure 1.3c is a cylinder emptied of a cone on its opposite faces. It is not a manifold since point v is not homeomorphic to a ball. However, the object is a quasi-manifold: slicing it along the blue plane yields two volumes which are manifold cells. This quasi-manifold property defines a *topological consistency* on the objects that should be preserved through transformations.

1.3 Embedding of the topological structure

In topology-based geometric modeling, the complete description of an object relies on the *embedding* of the topological subdivision into a geometric space [121]. Typically, the embedding of

an i-cell maps it to a space homeomorphic to a ball of dimension i. For a 3D object, we map each 0-cell to a point, each 1-cell to a curve, each 2-cell to a face, and each 3-cell to a volume. We are usually interested in continuous mapping where incident (resp. adjacent) cells are mapped to incident (resp. adjacent) balls. For instance, we map two adjacent topological faces incident to the same topological edge, i.e., 2-cells sharing a 1-subcell, to adjacent geometric faces incident to the same geometric edge. For formal reasoning about surfaces [28, 45], embedding the topological structure into a Euclidean space usually provides all the tools needed.

However, other geometric modeling applications may require other kinds of geometric information. For instance, we may want to add colors for rendering, physical values for simulations, or semantic information for explanation purposes. Therefore, a topological structure may be embedded in several different geometric spaces. Within a geometric space, all cells may not have to be embedded. A piecewise linear embedding might be enough if we only want to display the object. This embedding maps a topological vertex to a point in the 3D Euclidean space, an edge to a line segment, a face to a polygon, and a volume to a polyhedron. Such a piecewise embedding can be described entirely from the positions of the topological vertices if we account for its continuity. The line segments join two points, the polygons describe sequences of segments, and the polyhedrons consist of volumes bordered by polygons. For this representation, the positions of the geometric points associated with the topological vertices are enough to retrieve the complete object. Computer graphics tend to store all values on the vertices of triangular meshes. For instance, texturing maps each mesh vertex to coordinate in a 2D image. Then, the actual value within each triangle is computed by interpolation on the image. However, other applications may require that values are stored on edges, faces, or volumes. For instance, spring-mass systems in physical modeling may require providing spring constants on the edges [17]. Similarly, the simulation of protein traversals in biological cells may exploit permeability values on faces [143]. In its most general setting, embedding a topological structure can be described as adding geometric information on a cell basis.

1.4 Modeling operations

1.4.1 How to design modeling operations?

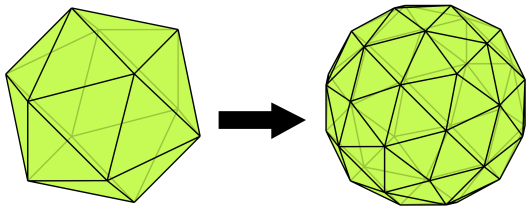


Figure 1.6: Application of the Butterfly [51] subdivision to an icosahedron.

The data structure presented in the previous section allows for storing geometric objects. We still have to elucidate the modifications of these objects, i.e., the modeling operations. An example of modeling operation is given in Figure 1.6 Standard approaches to designing new modeling operations rely on manual development, i.e., the implementation of algorithms. Such algorithms directly modify the data structure to obtain transformations of the modeled object. In the context

of this thesis, we are interested in higher-level approaches that can support formal reasoning with the ambition to provide a description allowing for the inference of operations.

Formal-rule languages have been used for twenty-five years in the context of geometric modeling and are usually referred to as procedural modeling techniques. Typically, these techniques avoid manually editing objects, proving fruitful modeling for regular objects, i.e., objects with many repetitions of sub-patterns. Thus, procedural modeling techniques have been exploited to generate plants [148], terrain [164], buildings [132], or cities [136]. From another perspective, using rules reduces the implementation and debugging effort. Indeed, once the rule application engine has been defined, the design of new modeling operations is reduced to the design of new rules.

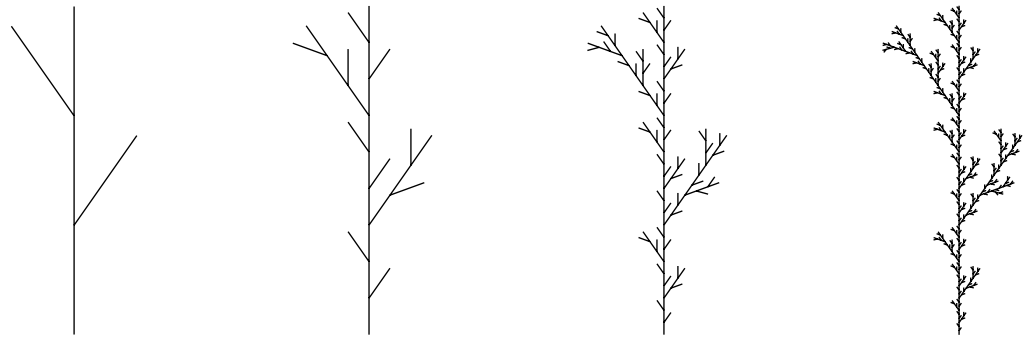


Figure 1.7: First iterations of the L-system F → F[+F]F[−F]F (a) first iteration corresponding to the word F[+F]F[−F]F, (b) second iteration corresponding to the word F[+F]F[−F]F[+F[+F]F[−F]F]F[+F]F[−F]F[−F[+F]F[−F]F]F[+F]F[−F]F, (c) third iteration, and (d) fourth iteration.

Lindenmayer systems, or L-systems, were introduced to describe plant cells' behavior and, in particular, their growth processes [122]. Formally, an L-system is a parallel rewriting system that builds objects by recursively applying production rules from an axiom word until a stop condition is met. The produced string is transformed into a geometric object via an interpretation [157, Part II Chap. 6 and Part III Chap. 5]. For instance, the L-system described by the axiom F and the single rule F → F[+F]F[−F]F yields the images of Figure 1.7 (Example taken from [147]).

Graph rewriting being an extension of string rewriting, the approach proposed in this dissertation can be seen as a generalization of L-systems to non-linear structures within the field of geometric modeling. As stated in the introduction, our motivation for working within a formal framework mostly comes from the idea that such a formalism will help us infer modeling operations. We now argue that a formal framework also eases the design of modeling operations while providing tools to study the consistency of the model through transformations.

1.4.2 Modeling operations from a software engineering perspective

In standard approaches to designing new modeling operations, conscientious people will prove the correctness of the algorithms with respect to a model. Then, if we assume a mathematically sound model, the produced object will satisfy some desirable properties, e.g., manifoldness. For instance, [44] uses combinatorial maps to represent oriented (quasi)-manifolds. The paper's key idea is to abstract modeling operations in a query-and-replace framework, similar to 'Find' and 'Replace' in a text editor. The authors demonstrate that their algorithms produce combinatorial

maps assuming valid combinatorial maps as input. However, the proof must be repeated whenever a new algorithm is designed. Proof of correctness can also be obtained with the help of a prover like Coq [150].

On the other hand, formal languages usually rely on a small set of elementary rules that are shown to be coherent. Any other transformation is decomposed in a sequence of elementary rules and therefore preserves the model's consistency. In this case, the rules are tailored to the applicative domain and are not necessarily transposable to other domains. With our ambition to provide a domain-independent solution, we aim to establish a definition of modeling operations and show that any such operations preserve the axiomatization of a well-defined model. Phrased like this, we only moved the issue from proving that an algorithm produces valid objects to proving that a transformation is a valid modeling operation. Although it can already be a simplification if the properties are easier to prove, some proofs still need to be carried out. In order to remove the need for any such proof, we propose a language to express modeling operations. Proving the correctness of the language can be done once, and then any operation designed with the language is guaranteed to produce only well-formed objects.

Summary of the chapter's contributions

In this chapter, we have presented some representations of objects commonly used in geometric modeling with an emphasis on combinatorial maps used in topology-based geometric modeling. These combinatorial maps represent the topological and geometric content of the object separately. The represented objects are (quasi-)manifolds, meaning that modeling operations should preserve the topological consistency of the model. The geometric content is handled via embedding functions that provide values to the topological cells. Modeling operations represent the core of this dissertation. They can be described differently based on the actual representation of the modified object, although most approaches derive algorithms working directly at the data structure level. We argue that a higher-level description simplifies the processes of both checking the consistency of the produced object, developed thoroughly in Part I, and inferring operations from examples, which is the topic of Part II.

Part I

Formalization of geometric modeling operations as graph transformation rules

Chapter 2

Graph transformations

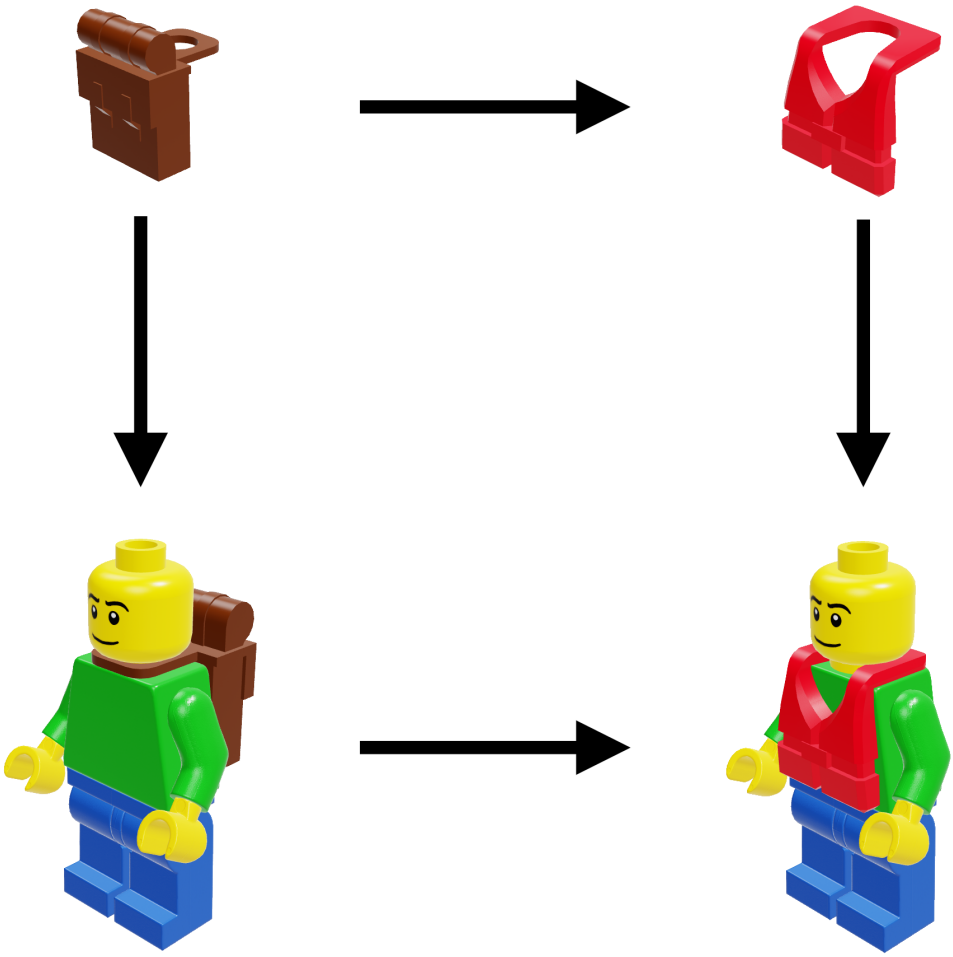


Figure 2.1: We learned rewriting at a very young age.

Personal note on the chapter

Graph transformation can either be explained with abstract notions stemming from category theory or via the help of sets and an element-based description of pushouts called the gluing construction [55, 60, 111, 99]. I lived happily in this set-based point of view for quite some time until I realized the tools I had at my disposal were not suited for the tasks. Therefore, I read some lecture notes from G. Huet [104], which provide an equational approach to category theory, and the first chapters of the textbook from M. Barr and C. Wells [7] without fully understanding what was at stake. I finally found a series of online lectures by B. Milewski,[1] called Category Theory, addressed to programmers. Although the lectures do not always provide all the mathematical formalism, B. Milewski gives a lot of code-based examples and motivations, which helped me greatly to get better intuition about some of the notions. Most of my understanding of category theory actually comes from these lectures. Nevertheless, to help with the mathematical foundations of category theory, I relied on the book from T. Leinster [118].

Regarding graph transformations, the tutorial [55] and introduction [60] articles from H. Ehrig provide an easy way into the theory, assuming knowledge of string rewriting theory. More recently, the book by R. Heckel and G. Taentzer [99] presents a more up-to-date description of graph rewriting for non-experts of the domain. Formal definitions and constructions presented in this chapter mostly come from the book by H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer [57].

Contents

[1] Available on Youtube https://www.youtube.com/user/DrBartosz/playlists: Category Theory, Category Theory II and Category Theory III.

Graph transformations were introduced to generalize Chomsky grammars to non-linear structures [55]. The algebraic approach to graph transformations uses rewriting rules where the left-hand and right-hand sides are graphs. Intuitively, a rewriting rule looks like $A \to B$ and stands for the modification of A into B within a bigger context C. In the context of graph transformations, A, B, and C are graphs, which raises several questions. For instance, what does the modification of A into B means? Is it sound? How do we identify A in C? Once we replace A by B in C, do we still have a graph? These questions have been answered first using the notion of graph gluing via graph morphisms. Graph gluing can be generalized with constructions from category theory, namely single-pushout or double-pushout [60], permitting the rewriting of more general structures.

In Section 2.1, we introduce the main abstract notions from category theory that will be needed throughout this manuscript. Categories related to graphs presented in Section 2.2 constitute the critical structures of this dissertation. We will describe geometric objects as graphs to formalize modeling operations as graph rewriting rules. We present the double-pushout approach to graph rewriting in Section 2.3 from the axiomatization of adhesive categories. Finally, we discuss tools and applications of graph transformations with an emphasis on geometric-related approaches in Section 2.4.

2.1 Category theory

Category theory is a branch of abstract algebra that studies mathematical structures and their relations. This theory has seen quite an important echo in theoretical computer science [24, 7].

A key concept of category theory is *universal properties*. Universal properties describe an object by explaining its relations with all other objects of the universe in which it lives. The universal properties we will need for this manuscript are pushouts, pullbacks, products, and terminal objects, but first, we define categories.

Most of the elements presented in this section come from [118].

2.1.1 Categories

A category describes objects in relation with is each other. These relations are called morphisms, maps, or arrows.

Definition 3 (Category [118])**.** *A* category $\mathscr{A}$ *consists of:*

- *a collection* $\mathscr{O}(\mathscr{A})$ *of* objects*;*
- *for each* $A, B \in \mathscr{O}(\mathscr{A})$*, a set* $\mathscr{A}(A, B)$ *of* morphisms *(also called* arrows *or* maps*) from* A *to* B*;*
- *for each* $A, B, C \in \mathscr{O}(\mathscr{A})$*, a function*

$$\begin{cases} \mathscr{A}(B,C) \times \mathscr{A}(A,B) & \to & \mathscr{A}(A,C) \\ (g,f) & \mapsto & g \circ f \end{cases}$$

  called composition*, satisfying the following* associativity law*: for each* $f \in \mathscr{A}(A,B)$*,* $g \in \mathscr{A}(B,C)$ *and* $h \in \mathscr{A}(C,D)$*, we have* $(h \circ g) \circ f = h \circ (g \circ f)$*;*

- *for each* $A \in \mathscr{O}(\mathscr{A})$*, an element* 1_A *of* $\mathscr{A}(A,A)$*, called the* identity *on* A*, satisfying the following* identity law*: for each* $f \in \mathscr{A}(A,B)$*, we have* $f \circ 1_A = f = 1_B \circ f$*.*

Notation 1. *We write* $A \in \mathscr{A}$ *for* $A \in \mathscr{O}(\mathscr{A})$ *and* $f\colon A \to B$ *or* $A \xrightarrow{f} B$ *for* $f \in \mathscr{A}(A,B)$.

Example 1 (Empty and trivial categories)**.** There is a category $\emptyset$ with no object and no morphism called the *empty category*. There is a category $*$ with one object and its identity called the *trivial category*.

Example 2 (Category of sets)**.** The category **Set** has sets for objects and functions between sets as morphisms. In **Set**, the identity morphism is the identity function, and the composition of morphisms is the usual composition of functions.

Example 3 (Preorders as categories)**.** Given a set S, a preorder $\leq$ on S is a reflexive (for all x in S, $x \leq x$) and transitive (for all x, y, and z in S, if $x \leq y$ and $y \leq z$ then $x \leq z$) relation on S. A S with a preorder $\leq$ forms a preordered set $(A, \leq)$.

Any preordered set $(A, \leq)$ can be regarded as a category $\mathscr{S}$ whose objects are the elements of S and such that there is a morphism $x \to y$ if and only if $x \leq y$. The reflexivity gives the identities on the objects, and the transitivity provides the composition of morphisms.

Reversely, if the objects of a category $\mathscr{A}$ form a set and for any two objects A and B in $\mathscr{A}$, the set $\mathscr{A}(A,B)$ of morphisms from A to B contains at most one morphism, then $\mathscr{A}$ describe a preordered set. Indeed, we can write $A \leq B$ whenever there is a morphism $A \to B$. Then, the existence of identity morphisms ensures the relation's reflexivity, while morphisms' composition guarantees its transitivity.

Many mathematical structures form categories, e.g., groups with group morphisms, rings with ring morphisms, and topological spaces with continuous maps. Additionally, some structures, such as preorders or monoids, can be seen as categories. In this dissertation, we are interested in categories related to graphs defined in Section 2.2.

Definition 4 (Isomorphism [118])**.** *A morphism* $f\colon A \to B$ *in a category* $\mathscr{A}$ *is an* isomorphism *if a morphism* $g\colon B \to A$ *exists in* $\mathscr{A}$ *such that* $g \circ f = 1_A$ *and* $f \circ g = 1_B$.

In this case, g *is called the* inverse *of* f*, while* A *and* B *are said to be* isomorphic.

Categories allow describing objects, morphisms, and their behavior through *commutative diagrams*. Intuitively, a diagram in a category $\mathscr{A}$ consists of objects connected with morphisms, which are often drawn, such as

$$\begin{array}{ccc} B & \xleftarrow{f} & A \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle i} \\ C & \xrightarrow[h]{} & D \end{array}$$

where the morphisms are drawn as labeled arrows. This diagram *commutes* if $h \circ g \circ f = i$. Generally speaking, a diagram commutes if any two paths between objects X and Y yield equal morphisms by composition along the paths. We will be particularly interested in diagrams of the following form, called *commutative squares*.

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle f'}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[g']{} & D \end{array}$$

2.1.2 Functors and the category of categories

Categories being mathematical objects themselves, category theory leads to asking what a sensible notion of morphism between these categories is. The notion of morphism between categories is called *functor*.

Definition 5 (Functor [118])**.** *A* functor *between two categories $\mathscr{A}$ and $\mathscr{B}$, written* $\mathrm{F}\colon \mathscr{A} \to \mathscr{B}$*, consists of:*

- *a function* $\mathscr{O}(\mathscr{A}) \to \mathscr{O}(\mathscr{B})$*, written* $\mathrm{A} \mapsto \mathrm{F}(\mathrm{A})$*;*
- *for each* $\mathrm{A}, \mathrm{A}' \in \mathscr{A}$*, a function* $\mathscr{A}(\mathrm{A}, \mathrm{A}') \to \mathscr{B}(\mathrm{F}(\mathrm{A}), \mathrm{F}(\mathrm{A}'))$*, written* $f \mapsto \mathrm{F}(f)$*,*

satisfying the following axioms:

- $\mathrm{F}(f' \circ f) = \mathrm{F}(f') \circ \mathrm{F}(f)$*, for all* $f\colon \mathrm{A} \to \mathrm{A}'$ *and* $f'\colon \mathrm{A}' \to \mathrm{A}''$ *in* $\mathscr{A}$*;*
- $\mathrm{F}(1_{\mathrm{A}}) = 1_{\mathrm{F}(\mathrm{A})}$*, for all* $\mathrm{A} \in \mathscr{A}$*.*

Example 4 (Powerset functor)**.** The (covariant) powerset functor $\mathrm{P}\colon$ **Set** $\to$ **Set** maps a set X to the set of all its subset and a function $f\colon \mathrm{X} \to \mathrm{Y}$ to its direct image function $\mathrm{P}(f)$, i.e., for all U in $\mathrm{P}(\mathrm{X})$, $\mathrm{P}(f)(\mathrm{U}) = \{f(x) \mid x \in \mathrm{U}\}$.

Example 5 (Order-preserving maps)**.** Given two preordered set $(\mathrm{A}, \leq_{\mathrm{A}})$ and $(\mathrm{B}, \leq_{\mathrm{B}})$ viewed as categories $\mathscr{A}$ and $\mathscr{B}$ (in the sense of Example 3), the functors $\mathscr{A} \to \mathscr{B}$ are the order-preserving maps, i.e., the functions $f\colon \mathrm{A} \to \mathrm{B}$ such that for all a and a' in A, $a \leq_{\mathrm{A}} a'$ implies $f(a) \leq_{\mathrm{B}} f(a')$.

Example 6 (Category of categories)**.** There is a category **Cat** whose objects are the small categories (where the collections of objects are sets), and morphisms are the functors between small categories.

2.1.3 Initial and terminal objects

We next present our first universal constructions, namely initial and terminal objects.

Definition 6 (Initial and terminal objects [118])**.** *An object $\varnothing_{\mathscr{A}}$ in a category $\mathscr{A}$ is* initial *if for every $\mathrm{A} \in \mathscr{A}$, there is exactly one morphism $\varnothing_{\mathscr{A}} \to \mathrm{A}$. By duality, an object $1_{\mathscr{A}}$ in a category $\mathscr{A}$ is* terminal *if for every $\mathrm{A} \in \mathscr{A}$, there is exactly one morphism $\mathrm{A} \to 1_{\mathscr{A}}$.*

Duality plays a crucial role in category theory, meaning that any definition or theorem (and proof) has a *dual* version obtained by reversing all the arrows.

Example 7 (Initial and terminal objects in a discrete category)**.** A category may not have initial and terminal objects. A *discrete category* is a category whose only morphisms are the identities on the object. Therefore, the only discrete category with terminal and initial objects is the trivial category $*$.

Example 8 (Initial and terminal objects in **Set**)**.** The empty set $\emptyset$ is initial in **Set** and all singletons (sets containing only one element) are terminal in **Set**.

Example 9 (Initial and terminal objects in **Cat**)**.** The empty category $\emptyset$ is initial, and the trivial category $*$ is terminal in **Cat**.

Categories may not have initial and terminal objects, but if they exist, they are unique *up to unique isomorphism.*

Proposition 1. *If $\varnothing$ and $\varnothing'$ are initial objects in a category $\mathscr{A}$, then there is a unique isomorphism $\varnothing \to \varnothing'$.*

Therefore, we can speak of *the* initial object of a category $\mathscr{A}$, hence the notation $\varnothing_{\mathscr{A}}$. By duality, the result holds for terminal objects.

Notation 2. *If a category $\mathscr{A}$ admits a terminal object, we write $!_A$ for the unique morphism $A \to \mathbb{1}_{\mathscr{A}}$, where A is an object of $\mathscr{A}$.*

We will mostly be interested in terminal objects.

2.1.4 Slice and comma categories

We now introduce the notion of slice and comma categories to define typed graphs and undirected graphs.

Definition 7 (Slice categories)**.** *Let A be an object of a category $\mathscr{A}$. The* slice category *of $\mathscr{A}$ over A, written $\mathscr{A}/A$ is the category such that:*

- *the objects of $\mathscr{O}(\mathscr{A}/A)$ are the morphisms $X \to A$ in $\mathscr{A}$,*
- *for any two morphisms $f : X \to A$ and $f' : X' \to A$ in $\mathscr{A}$, the set of morphisms from f to f' in $\mathscr{A}/A$ are the morphisms $g : X \to X'$ such that $f' \circ g = f$, i.e., such that the following diagram commutes.*

$$\begin{array}{ccc} X & \xrightarrow{\quad g \quad} & X' \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle f'} & \\ & A & \end{array}$$

Definition 8 (Comma category [118])**.** *Given categories and functors $\mathscr{A} \xrightarrow{P} \mathscr{C} \xleftarrow{Q} \mathscr{B}$ the* comma category $(P \downarrow Q)$ *consists of:*

- *triples (A, h, B) for objects with $A \in \mathscr{A}$, $B \in \mathscr{B}$, and $h : P(A) \to Q(B)$ in $\mathscr{C}$;*
- *pairs of morphisms $(f : A \to A', g : B \to B')$ for the morphism $(A, h, B) \to (A', h', B')$ such that the following square commutes.*

$$\begin{array}{ccc} P(A) & \xrightarrow{P(f)} & P(A') \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ Q(B) & \xrightarrow[Q(g)]{} & Q(B') \end{array}$$

Slice categories are particular kinds of comma categories. Indeed, given an object A in a category $\mathscr{A}$, the slice category $\mathscr{A}/\mathrm{A}$ is the comma category $(1_{\mathscr{A}} \downarrow \mathrm{A})$, where $1_{\mathscr{A}} : \mathscr{A} \to \mathscr{A}$ is the identity functor on the category $\mathscr{A}$ and the object A is identified with the unique functor $\mathrm{A} : * \to \mathscr{A}$ that maps the unique object of $*$ to A. Therefore, we might further specify objects in a slice category $\mathscr{A}/\mathrm{A}$ as a pair (X, f) rather than as a morphism $f : \mathrm{X} \to \mathrm{A}$, which allows talking about the object X in $\mathscr{A}$ more naturally.

Lemma 1. *In a slice category $\mathscr{A}/\mathrm{A}$, the identify morphism 1_{A} is terminal in $\mathscr{A}/\mathrm{A}$, although we might also say that* A *is terminal in $\mathscr{A}/\mathrm{A}$.*

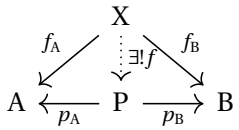
2.1.5 Limits and colimits

To study graph rewriting, we need a few more notions from category theory, related to *limits* and *colimits*. (Co)limits describe relationships within categories, unifying many concepts and constructions in mathematics. Products, pullbacks, and pushouts are the three notions of (co)limit that we will need.

Notation 3 (Span and cospan)**.** *A diagram of the form* $\mathrm{A} \xleftarrow{g} \mathrm{C} \xrightarrow{h} \mathrm{B}$ *is called a* span*, while its dual* $\mathrm{A} \xrightarrow{g} \mathrm{C} \xleftarrow{h} \mathrm{B}$ *is called a* cospan*.*

Spans and cospans help define products.

Definition 9 (Product (adapted from [118]))**.** *Let $\mathscr{A}$ be a category and* $\mathrm{A}, \mathrm{B} \in \mathscr{A}$*. A* product *of* A *and* B *consists of a span* $\mathrm{A} \xleftarrow{p_{\mathrm{A}}} \mathrm{P} \xrightarrow{p_{\mathrm{B}}} \mathrm{B}$ *with the property that for all spans* $\mathrm{A} \xleftarrow{f_{\mathrm{A}}} \mathrm{X} \xrightarrow{f_{\mathrm{B}}} \mathrm{B}$ *in $\mathscr{A}$, there exists a unique*[2] *morphism $f : \mathrm{X} \to \mathrm{P}$ such that the following diagram commutes.*



The morphisms p_{A} and p_{B} are called the projections*.*

Example 10 (Products in a discrete category)**.** Products may not exist for all pairs of objects. For instance, two distinct objects in a discrete category do not admit a product.

When two objects A and B admit a product, it is unique up to isomorphism, justifying to talk about *the* product of A and B.

Example 11 (Products in **Set**)**.** In **Set**, the product of two sets A and B is the Cartesian product $\mathrm{A} \times \mathrm{B}$ equipped with the usual projections. Therefore, **Set** is said to be with *all products*, as any pair of sets admits a product.

Even though the product of A and B consists of the object P and the projections p_{A} and p_{B}, we will often talk about P alone as the product of A and B and write it $\mathrm{A} \times \mathrm{B}$.

Pullbacks can be seen as products with additional information where the two objects share morphisms to the same third object.

[2] The existence of a unique morphism in a diagram is denoted with a dashed arrow and the quantifier $\exists!$.

Definition 10 (Pullback (adapted from [118]))**.** *Let $\mathscr{A}$ be a category, and consider the cospan* $A \xrightarrow{g} C \xleftarrow{h} B$ *in* $\mathscr{A}$*. A* pullback *of this cospan is a span* $A \xleftarrow{p_A} P \xrightarrow{p_B} B$ *such that*

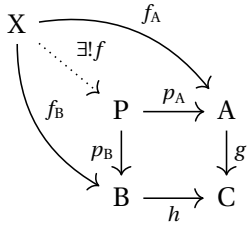
- *the following square commutes :*

$$\begin{array}{ccc} P & \xrightarrow{p_A} & A \\ {\scriptstyle p_B}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[h]{} & C \end{array} \tag{2.1}$$

- *for any commutative square*

$$\begin{array}{ccc} X & \xrightarrow{f_A} & A \\ {\scriptstyle f_B}\downarrow & & \downarrow{\scriptstyle g} \\ B & \xrightarrow[h]{} & C \end{array}$$

in $\mathscr{A}$*, there is a unique map* $f : X \to P$ *such that*



commutes.

Again, a pullback needs not to exist, but if it exists, it is unique up to isomorphism, which enables talking about *the* pullback. The square (2.1) is called *pullback square,* but we will sometimes talk about the pullback to refer to its square. Pullbacks are also called *fibred products.*

Lemma 2. *The pullbacks of* $A \xrightarrow{!_A} 1_{\mathscr{A}} \xleftarrow{!_B} B$ *in a category* $\mathscr{A}$*, provided it exists, is the product of* A *and* B*, the unique morphism* f *in Definition 9 is simply* $!_{A\times B}$*.*

Example 12 (Pullback in **Set**)**.** The pullback of $A \xrightarrow{g} C \xleftarrow{h} B$ in **Set**, is the set $P = \{(a,b) \in A \times B \mid g(a) = h(b)\}$ with projections $p_A(a,b) = a$ and $p_B(a,b) = b$.

The inverse image of a function is computed as a pullback in **Set**. If f is a function $X \to Y$ and Z is a subset of Y, meaning there is a canonical injection $Z \hookrightarrow Y$, then $f^{-1}(Z)$ is the pullbacks (object) of $X \xrightarrow{f} Y \hookleftarrow Z$.

Similarly, the intersection of two sets can be obtained as a pullback.

The dual notion of pullback is pushout, where all morphisms in the definition of a pushout are reversed.

Definition 11 (Pushout)**.** *Let $\mathscr{A}$ be a category. The* pushout *of* $A \xleftarrow{g} C \xrightarrow{h} B$ *is a cospan* $A \xrightarrow{p_A} P \xleftarrow{p_B} B$ *that makes the square*

$$\begin{array}{ccc} C & \xrightarrow{g} & A \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle p_A} \\ B & \xrightarrow[p_B]{} & P \end{array}$$

commutes while being universal with this property. The morphisms p_A *and* p_B *are called the* coprojections *of the pushout.*

Example 13 (Pushouts in **Set**)**.** The pushout of $A \xleftarrow{g} C \xrightarrow{h} B$ in **Set** is $P = (A \cup B)/\sim$, where $\sim$ is the equivalence relation on C such that $g(c) \sim h(c)$ for all $c \in C$. The coprojection $p_A : A \to P$ (resp. $p_B : B \to P$) maps $a \in A$ (resp. $b \in B$) to its equivalence class in P.

For instance, if A and B are subsets of the same set X, the following square is a pushout in **Set**.

$$\begin{array}{ccc} A \cap B & \longrightarrow & A \\ \downarrow & & \downarrow \\ B & \longrightarrow & A \cup B \end{array}$$

Note that the square is also a pullback.

Pushouts share the same properties as pullbacks by duality.

2.1.6 Mono and epi

Before moving on to graph transformations, we provide one last notion and its dual counterpart. The notions of monos and epis generalize injectivity and surjectivity in category theory.

Definition 12 (Monic (adapted from [118]))**.** *A morphism* $f : A \to B$ *in* $\mathscr{A}$ *is* monic*, written* $f : A \hookrightarrow B$*, if it is left-cancellable, i.e., for all object* $X \in \mathscr{A}$ *and all pair of morphisms* $g, g' : X \to A$*,* $f \circ g = f \circ g' \implies g = g'$*.*

A monic morphism is also called a monomorphism *or simply a* mono*.*

We could have defined monos straight away when talking about morphisms, but we present them alongside limits because of the following relation between monos and pullbacks.

Lemma 3. *A morphism* $f : A \to B$ *is a mono if and only if the following square is a pullback.*

$$\begin{array}{ccc} A & \xrightarrow{1_A} & A \\ {\scriptstyle 1_A}\downarrow & & \downarrow{\scriptstyle f} \\ A & \xrightarrow{f} & B \end{array}$$

The dual notion of a monomorphism is an epimorphism.

Definition 13 (Epic (adapted from [118]))**.** *A morphism* $f : A \to B$ *in* $\mathscr{A}$ *is* epic*, if it is right-cancellable, i.e., for all object* $X \in \mathscr{A}$ *and all pair of morphisms* $g, g' : B \to X$*,* $g \circ f = g' \circ f \implies g = g'$*.*

An epic morphism is also called a epimorphism *or simply an* epi*.*

By duality, we have the same lemma as Lemma 3 with a pushout.

Example 14 (Monos and epis in **Set**)**.** In **Set**, the monomorphisms are the injections, and the epimorphisms are the surjections.

An isomorphism is both epic and monic.

Lemma 4. *The classes of monomorphisms, epimorphisms, and isomorphisms are closed by composition, i.e., the composition of two monomorphisms (resp. epimorphisms, isomorphisms) is a monomorphism (reps. epimorphism, isomorphism)*

In relation to combinatorics, we will also have an interest in involutions. There are several ways to think about an involution. With the terminology already introduced, we can define an involution as follows.

Definition 14 (Involution)**.** *An* involution *is an isomorphism from an object to itself which is its own inverse, i.e.,* $i : \mathrm{A} \to \mathrm{A}$ *in* $\mathscr{A}$ *is an involution if* $i \circ i = 1_{\mathrm{A}}$.

Example 15 (Involutions in **Set** and fix points)**.** Consider the set $\mathbb{R}$ of real numbers, $f : \mathbb{R} \to \mathbb{R}; x \mapsto -x$ is an involution. For a set $\mathrm{X} \in$ **Set**, a *fix point* of an involution $i : \mathrm{X} \to \mathrm{X}$ is an element $x \in \mathrm{X}$ such that $i(x) = x$.

2.2 Categories of graphs

Graphs are ubiquitous in computer science, and their definition varies between communities, e.g., algorithmics, combinatorics, and category theory. The notion of graphs in category theory is close to the definition of a category (to the point that [7] defines categories as particular kinds of graphs). This section presents the category **Graph** of graphs, also called multigraphs with parallel arcs and loops. We also discuss some variations of graph-based categories.

2.2.1 The category of graph and graph morphisms

A graph is a collection of objects, called *nodes* or *vertices*, linked with arrows, called *arcs* or *edges*. The lack of constraint on the arcs means that a graph is a directed graph with parallel arcs and loops. The source and target of each arc in a graph are identified with two special functions (s for source and t for target), such that a graph can be described by the following diagram in the category **Set**:

$$\mathrm{E} \underset{t}{\overset{s}{\rightrightarrows}} \mathrm{V} \tag{2.2}$$

where V is the set of nodes and E is the set of arcs.

Definition 15 (Graph [57])**.** *A* graph $\mathrm{G} = (\mathrm{V_G}, \mathrm{E_G}, s_{\mathrm{G}}, t_{\mathrm{G}})$ *consists of a set of* nodes $\mathrm{V_G}$*, a set of* arcs $\mathrm{E_G}$*, and two functions* $s_{\mathrm{G}}, t_{\mathrm{G}} : \mathrm{E_G} \to \mathrm{V_G}$*, the* source *and* target *functions.*

The subscripts $_{\mathrm{G}}$ will be omitted when there is no ambiguity about the graph. A *path* is a sequence $e_1 \dots e_n$ of arcs such that $t_{\mathrm{G}}(e_k) = s_{\mathrm{G}}(e_{k+1})$ for all $1 \leq k < n$. If $s_{\mathrm{G}}(e_1) = t_{\mathrm{G}}(e_n)$ then the path is a *cycle*. An arc $e \in \mathrm{E}$ is said to be *incident* to the nodes $s(e)$ and $t(e)$. The arc e is *non-oriented* if it has a *reverse arc* $e' \in \mathrm{E}$, i.e., there exists an arc e' such that $s(e') = t(e)$ and $t(e') = s(e)$. When the reverse arc is unique, we write $e' = e^{-1}$.

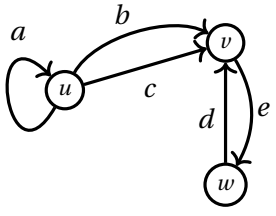


Figure 2.2: A graph.

Example 16 (A graph)**.** Figure 2.2 displays the graph $G = (\{u, v, w\}, \{a, b, c, d, e\}, s, t)$ with source and target function $s, t : E \to V$ defined as follows.

$$s: \begin{cases} a, b, c & \mapsto & u \\ e & \mapsto & v \\ d & \mapsto & w \end{cases} \quad \text{and } t: \begin{cases} a & \mapsto & u \\ b, c, d & \mapsto & v \\ e & \mapsto & w \end{cases}$$

The visual representation of the graph in Figure 2.2 contains the same information as the full specification from Example 16. Therefore, we can define graphs from their visual representation, e.g., call G the graph of Figure 2.2 without providing a plain-text definition.

To define the category of graphs, we still need a notion of graph morphism.

Definition 16 (Graph morphism [57])**.** *A* graph morphism $m = (m_V, m_E) : G \to H$ *consists of two functions* $m_V : V_G \to V_H$ *and* $m_E : E_G \to E_H$ *that preserve sources and targets, i.e., such that* $s_H \circ m_E = m_v \circ s_G$ *and* $t_H \circ m_E = m_v \circ t_G$.

From the diagrammatic definition (see diagram 2.2) of graphs in **Set**, a graph morphism $m = (m_V, m_E) : G \to H$ corresponds to the two following commutative diagrams.

$$\begin{array}{ccc} E_G & \xrightarrow{s_G} & V_G \\ {\scriptstyle m_E}\downarrow & & \downarrow{\scriptstyle m_V} \\ E_H & \xrightarrow[s_H]{} & V_H \end{array} \qquad \begin{array}{ccc} E_G & \xrightarrow{t_G} & V_G \\ {\scriptstyle m_E}\downarrow & & \downarrow{\scriptstyle m_V} \\ E_H & \xrightarrow[t_H]{} & V_H \end{array}$$

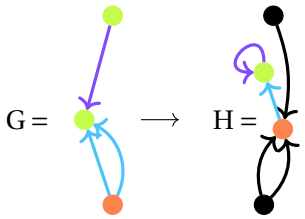


Figure 2.3: A graph morphism.

Example 17 (Graph morphism)**.** The morphism $G \to H$ of Figure 2.3 maps nodes and arcs by colors: the green nodes to the green node, the orange node to the orange node, the purple arc to the purple arc, and the blue arcs to the blue arc. The black elements in H are the image of no element from G. Note that preserving source and targets means that the purple arc is mapped to a loop.

Definition 17 (The category **Graph**)**.** *Graph and graph morphisms form the category* **Graph** *of* graphs.

The category **Graph** admits initial and terminal objects, products, pushouts, and pullbacks.

Proposition 2 (Initial and terminal objects in **Graph**)**.** *The empty graph* $G_\emptyset(\emptyset, \emptyset, 1_\emptyset, 1_\emptyset)$ *is initial in* **Graph**, *and the graph*  *is terminal in* **Graph**.

Proposition 3 (Products in **Graph**)**.** *In* **Graph**, *any two graphs* G *and* H *admits a product written* $G \times H$ *or* $G \otimes H$. *The* product *of* G *and* H, *sometimes called* cardinal product *[129] or* tensor product, *consists of:*

- $V_{G\times H} = V_G \times V_H$;
- $E_{G\times H} = E_G \times E_H$;
- $s_{G\times H} : E_{G\times H} \to V_{G\times H} : (e_G, e_H) \mapsto (s_G(e_G), s_H(e_H))$;
- $t_{G\times H} : E_{G\times H} \to V_{G\times H} : (e_G, e_H) \mapsto (t_G(e_G), t_H(e_H))$.

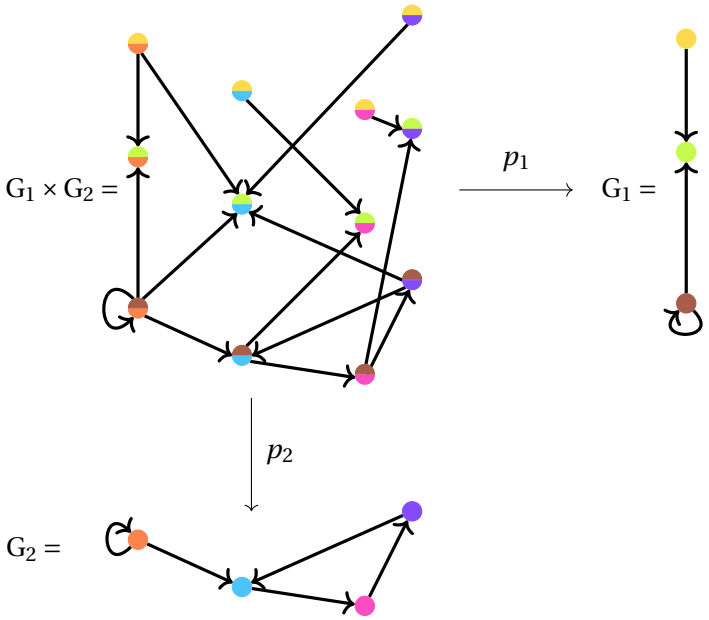
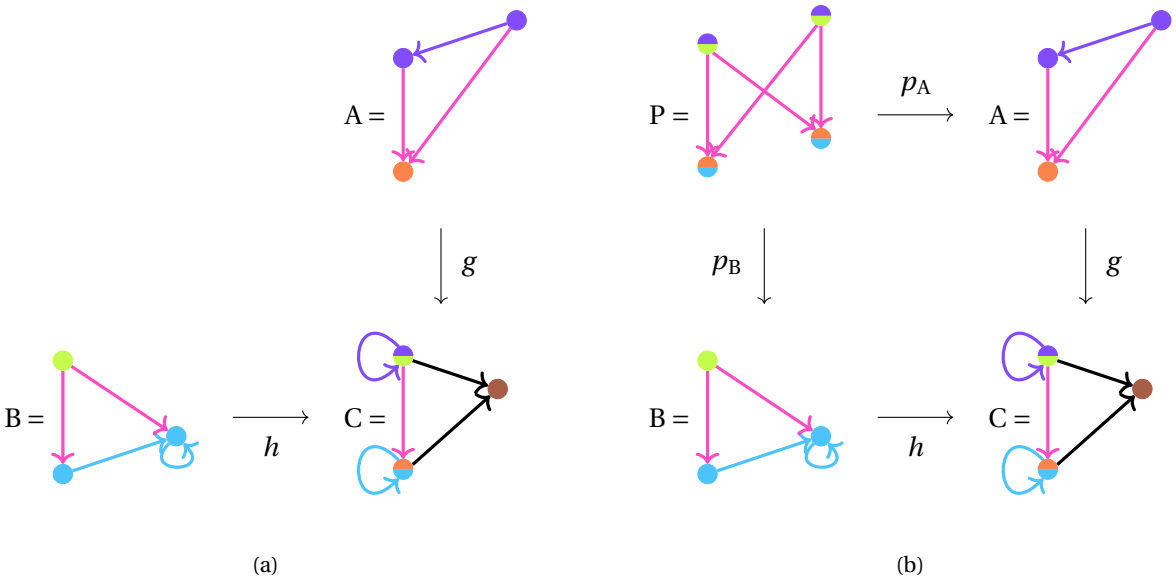


Figure 2.4: A graph product.

Example 18 (Product in **Graph**)**.** Figure 2.4 displays the product of G_1 and G_2. Nodes are bi-colored with the color of their projections to help visualize the result. The upper-half color of a node in $G_1 \times G_2$ indicates the image of the node in G_1, while the bottom-half color gives the node in G_2. For instance, the node ● in $G_1 \times G_2$ is projected on the node ● of G_1 by p_1 and on the node ● of G_2 by p_2. The projections map the arcs based on the source and target image. Reversely, the construction of $G_1 \times G_2$ from G_1 and G_2 is obtained componentwise on the set of nodes and arcs.

Pullbacks in **Graph** can be obtained componentwise from the pullbacks of the node and edge sets in **Set**.

Example 19 (Pullbacks in **Graph**)**.** Figure 2.5a describes a cospan of graphs where the morphisms are indicated by colors like in Example 18. For instance, the nodes ● of B are merged and mapped to the node ● of C. The pullback of $A \xrightarrow{g} C \xleftarrow{h} B$ is the span $A \xleftarrow{p_A} P \xrightarrow{p_B} B$ of Figure 2.5b. For nodes a in A and b in B such that $g(a) = h(b)$, there is a node in P. For example, the two nodes ● of A and the node ● of B share the node ● as image in C. Thus, P contains



(a) (b)

Figure 2.5: Pullback in **Graph**: (a) a span of graphs and (b) its pullback.

two nodes . Similarly, P has two nodes . The construction holds for the arcs. For instance, the pink arc in C admits two preimage in both A and B, which yields four arcs in P.

Proposition 4 (Monos and epis in **Graph**)**.** *In* **Graph**, *a morphism* $m = (m_V, m_E) : G \to H$ *is monic if and only if* m_V *and* m_E *are injective. Similarly,* m *is epic if and only if* m_V *and* m_E *are surjective.*

We will now discuss some specific cases of graphs.

2.2.2 Undirected graphs

Other notions of graphs also admit a categorical construction. For instance, if all arcs are considered non-oriented, we obtain the well-known notion of undirected graphs. A formal definition of these undirected graphs can be found in [161, p.13]: an undirected graph is a graph $G = (E, V, s, t)$ equipped with a fixed-point-free involution $i : E \to E$ such that $s \circ i = t$ and $t \circ i = s$. Undirected graphs can also be thought of by considering that an arc is a set of size 2 (or 1 for the case of loops) rather than a pair, leading to the following definition of undirected graphs.

Definition 18 (Undirected graph [11])**.** *Let* $P_{(1,2)} :$ **Set** $\to$ **Set** *be the functor*[3] *that maps a set* S *to its subsets of cardinality* 1 *or* 2*. The category of* undirected (multi)graph **uGraph** *has objects* G *which consists of*

- *a set of nodes* V_G,
- *a set of arcs* E_G,
- *an incidence function* $i_G : E_G \to P_{(1,2)}(V_G)$ *that provides, for each arc, its set of incident nodes,*

and morphisms $m : G \to H$ *constituted of functions* $m_V : V_G \to V_H$ *and* $m_E : E_G \to E_H$ *that commutes*

[3] The functor $P_{(1,2)}$ is called the restricted covariant power set functor.

with the incidence functions, i.e., such that the following diagram commutes.

$$\begin{array}{ccc} \mathrm{E_G} & \xrightarrow{m_\mathrm{E}} & \mathrm{E_H} \\ {\scriptstyle i_\mathrm{G}}\downarrow & & \downarrow{\scriptstyle i_\mathrm{H}} \\ \mathrm{P_{(1,2)}(V_G)} & \xrightarrow[\mathrm{P_{(1,2)}}(m_\mathrm{V})]{} & \mathrm{P_{(1,2)}(V_H)} \end{array}$$

In other words, **uGraph** *is the comma category* $(1_{\mathbf{Set}} \downarrow \mathrm{P}_{(1,2)})$ *where* $1_{\mathbf{Set}}$: **Set** → **Set** *is the identity functor on* **Set**.

For geometric modeling, we will manipulate two classes of graphs offering different representations of objects. The class of graphs associated with oriented maps is, as one can guess from the name, oriented, meaning that directed graphs are needed. In the case of generalized maps, undirected graphs would suffice. However, we will construct a unified framework for both models, so graphs need to be oriented. We will impose non-orientation a posteriori as a constraint on the graphs.

Similar to undirected graphs, the addition of decorations to a graph can be obtained via categorical constructions. We now present the addition of types to a graph while keeping the discussion of labels and attributes for the chapters where they will be needed.

2.2.3 Typed graphs

To design typed graphs, one could provide a set of types for the nodes and the arcs. Then, each node and each arc would have its own type. However, the relations between these types would be missing, i.e., we would not know which type of arc can link which type of node. Specifying these relations is, in essence, building a graph of types whose nodes are the types of nodes and whose arcs are the types of arcs linking the types of nodes appropriately. This graph, called the *type graph*, encodes both the types of the graph elements and their relations, effectively describing all valid graphs for these types, i.e., the typed graphs. This notion of typed graphs, first introduced in [35], exactly matches the notion of slice category. The key idea is to consider a distinguished type graph TG and to reason in the category **Graph**/TG.

Definition 19 (Typed graph [57])**.** *The category* $\mathbf{Graph}_\mathrm{TG}$ *of* typed graphs *(typed over a type graph* TG*) is the slice category* **Graph**/TG.

Since morphisms of typed graphs are commutative triangles, the type of an element (node or arc) cannot be modified by such morphisms. Hence, slicing the category **Graph** over a type graph ensures that we can easily preserve some of the graph structure while modifying it.

Example 20 (Typed graph)**.** We extend the color code from the example 17 to type the graphs G and H over the graph TG. The morphism G → H becomes a typed graph morphism, as illustrated in Figure 2.6 where the morphism associated with G and H in $\mathbf{Graph}_\mathrm{TG}$ is defined from the colors.

The properties of graphs hold by slicing. In particular [57, Chap. 2],

- $(\mathrm{TG}, 1_\mathrm{TG})$ is terminal in $\mathbf{Graph}_\mathrm{TG}$,
- $(\mathrm{G}_\emptyset, \mathrm{G}_\emptyset \to \mathrm{TG})$ is initial in $\mathbf{Graph}_\mathrm{TG}$,

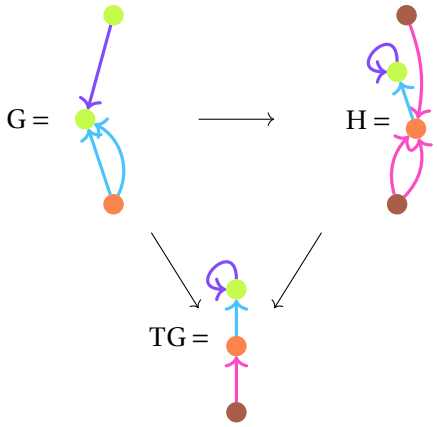


Figure 2.6: A morphism of typed graphs.

- monomorphisms of typed graphs are injective functions,
- epimorphisms of typed graphs are surjective functions,
- any two typed graphs admit a product,
- any span of typed graphs admits a pullback,
- any cospan of typed graphs admits a pushout,
- product, pullbacks, and pushouts can be constructed componentwise in $\mathbf{Graph}_{\mathrm{TG}}$.

Besides, products of typed graphs correspond to pullbacks of graphs.

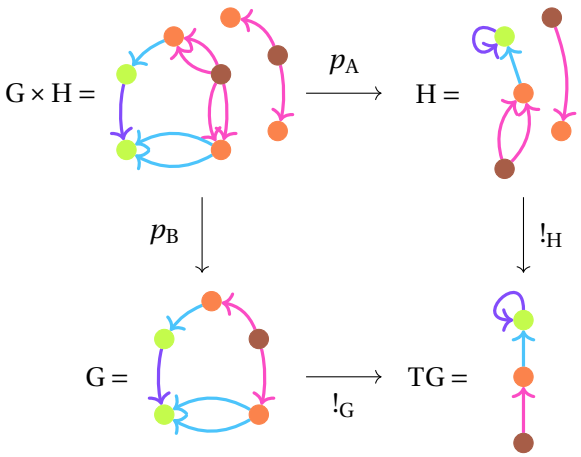


Figure 2.7: A product of typed graphs as a pullback.

Example 21 (Products in $\mathbf{Graph}_{\mathrm{TG}}$ as pullbacks in **Graph**)**.** In Figure 2.7, the graph G × H can be considered as the product of G and H in $\mathbf{Graph}_{\mathrm{TG}}$ or as pullback of G → TG ← H in **Graph**.

Any category $\mathbf{Graph}_{\mathrm{TG}}$ of graphs typed over TG defines a functor U : $\mathbf{Graph}_{\mathrm{TG}}\, to \mathbf{Graph}$

Example 22 (Forgetful functor on typed graphs)**.** Given a type graph TG, the category $\mathbf{Graph}_{\mathrm{TG}}$ of graphs typed over TG defines a functor U : $\mathbf{Graph}_{\mathrm{TG}} \to \mathbf{Graph}$ informally called the forgetful functor that forgets the typing. The forgetful functor maps a typed graph (G, m) to the graph G.

2.3 Graph rewriting

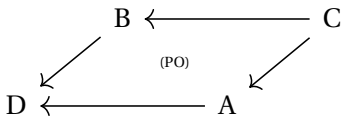
In this section, we present *graph rewriting*, emphasizing the framework we will use in the subsequent chapters, namely double-pushout rewriting (or DPO). We prefer double-pushout over single-pushout (or SPO) because in the SPO approach dangling edges after the deletion of the left-hand pattern are also deleted, which is less conservative. In other words, DPO eases the construction of conditions on rules for preserving constraints, as discussed in the following chapters. The main drawback is having to check the dangling condition, which is doable in a static manner for our specific class of graphs, as shown in [144].

We present double-pushout rewriting through the lens of adhesive categories. Variations of this notion exist, such as quasi-adhesivity, (vertical/horizontal) weak adhesivity, $\mathscr{M}$-adhesivity [116, 59] (see the summary done at the beginning of [11], or in [81]). Although the exact axioms of these properties vary, the general idea is that pushouts behave as in the category **Set**. The introduction of adhesivity provided a framework in which standard results hold, e.g., the local Church-Rosser, parallelism, and concurrency Theorem. For our needs, we exploit adhesivity only to ensure that DPO-rewriting is well-defined.

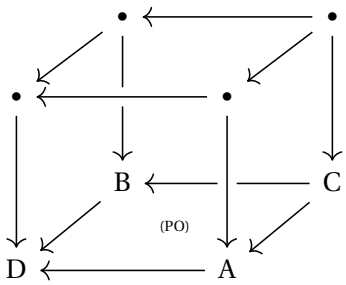
2.3.1 Adhesivity

We need some additional properties in a category to define a suitable notion of rewriting. Several such properties have been studied to generalize rewriting from the category **Graph** to more general categories. To minimally meet our requirements, a category should have pushouts along monomorphisms and products. The notion of adhesivity is commonly used in the graph rewriting community and fulfills these requirements.

Definition 20 (van Kampen square [115])**.** *Let $\mathscr{A}$ be a category. A pushout*



is a van Kampen square *if for all commutative cubes*



over the pushout square, such that the right and back faces are pullbacks, then the top face is a pushout if and only if the left and front faces are pullbacks.

With the help of van Kampen squares, we can define adhesivity.

Definition 21 (Adhesive category [115])**.** *A category $\mathscr{A}$ is* adhesive *if:*

- *$\mathscr{A}$ has pullbacks;*

Figure 2.8: A rule.

- *$\mathscr{A}$ has pushouts along monomorphisms, i.e., pushouts where at least one of the span morphisms is a monomorphism;*
- *pushouts along monomorphism are van Kampen squares.*

Example 23 (Graphs and typed graphs)**.** The category **Graph** is an adhesive category [115]. Besides, slicing preserves adhesivity, meaning that $\mathbf{Graph}_{\mathrm{TG}}$ is adhesive for any type graph TG.

In an adhesive category, the following lemmas hold.

Lemma 5. *Monomorphisms are stable under pushouts, i.e., the pushout of a span* $\mathrm{A} \hookleftarrow \mathrm{C} \to \mathrm{B}$ *is a cospan* $\mathrm{A} \to \mathrm{D} \hookleftarrow \mathrm{B}$*:*

$$\begin{array}{ccc} \mathrm{C} & \hookrightarrow & \mathrm{A} \\ \downarrow & & \downarrow \\ \mathrm{B} & \hookrightarrow & \mathrm{D} \end{array}$$

Lemma 6. *Pushouts along monomorphisms are also pullbacks.*

The Double-Pushout approach to graph rewriting relies on the existence of a pushout complement.

Definition 22 (Pushout complement)**.** *Let $\mathscr{A}$ be a category. The* pushout complement *of* $\mathrm{C} \to \mathrm{A} \to \mathrm{D}$ *is an object* B *together with a pair of morphisms* $\mathrm{C} \to \mathrm{B} \to \mathrm{D}$ *that makes the square*

$$\begin{array}{ccc} \mathrm{C} & \xrightarrow{g} & \mathrm{A} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle f} \\ \mathrm{B} & \xrightarrow[f']{} & \mathrm{D} \end{array}$$

a pushout.

In **Graph** and $\mathbf{Graph}_{\mathrm{TG}}$, if a pushout complement exists, it is unique up to isomorphism [57, Chap. 3, p.45]. In an adhesive category, the pushout complement of $\mathrm{A} \hookrightarrow \mathrm{B} \to \mathrm{C}$, called pushout complement of monos [115], is also unique up to isomorphism, provided it exists.

2.3.2 Graph transformations using double pushouts

Adhesivity ensures that the *Double-Pushout* (*DPO*) approach to graph rewriting is well-behaved. We choose DPO over other approaches for its conservatism and (relative) simplicity.

Definition 23 (Rule)**.** *A* rule *r is a span* $\mathrm{L} \hookleftarrow \mathrm{K} \hookrightarrow \mathrm{R}$ *of monomorphisms.*

The graphs L, R, and K are respectively called the *left-hand side*, *right-hand side*, and *interface* (or *kernel*) of the rule.

Example 24 (Rule)**.** Figure 2.8 illustrates a rule where the morphisms are given by the letters on the nodes, e.g., node a in K is mapped to node a in L and to node a in R. Intuitively, the rule replaces the path $abcd$ in L by a path aed in R.

Intuitively, applying a rule r to a graph G consists of three steps: matching L within G, deleting the matched elements that do not belong to K, and adding elements of R not in K. Matching the left-hand side of a rule within a graph is subject to the gluing condition that states the existence of a pushout complement.

Definition 24 (Match)**.** *Let* $\mathrm{L} \hookleftarrow \mathrm{K} \hookrightarrow \mathrm{R}$ *be a rule. A* match *for* r *is a morphism* $m\colon \mathrm{L} \to \mathrm{G}$*. The match* $m\colon \mathrm{L} \to \mathrm{G}$ *satisfies the* gluing condition *for* r *if there exists a pushout complement of* $\mathrm{K} \hookrightarrow \mathrm{L} \to \mathrm{G}$*.*

Matching the left-hand side L of a rule in a graph G is the first step toward transforming G. Still, we need to explain how to disconnect and reconnect graphs.

Definition 25 (Direct derivation)**.** *Let* G *and* H *be two graphs,* $r = \mathrm{L} \hookleftarrow \mathrm{K} \hookrightarrow \mathrm{R}$ *a rule, and* $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ *a match for* r*. The rule* r *transforms* G *into* H*, if there is a diagram*

$$\begin{array}{ccccc} \mathrm{L} & \hookleftarrow & \mathrm{K} & \hookrightarrow & \mathrm{R} \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow & (\mathrm{PO}) & \downarrow{\scriptstyle m'} \\ \mathrm{G} & \hookleftarrow & \mathrm{D} & \hookrightarrow & \mathrm{H} \end{array}$$

where both squares are pushouts. The direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ *exists and is unique (up to isomorphism) if* m *satisfies the gluing condition. The morphism* $m'\colon \mathrm{R} \hookrightarrow \mathrm{H}$ *is called the comatch of the derivation.*

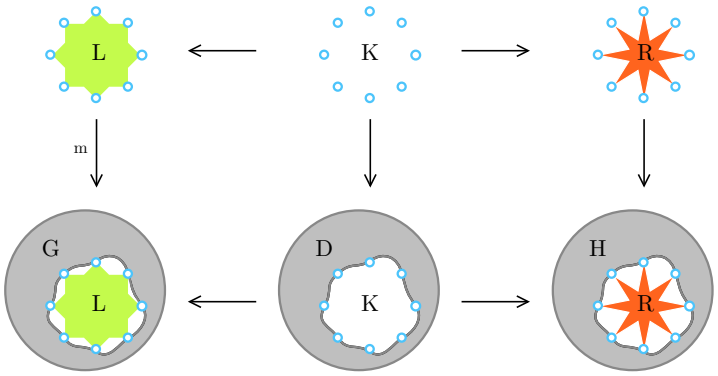


Figure 2.9: Intuitive description of a graph rewriting step in the DPO framework.

In Figure 2 (in the introduction), we presented an intuitive description of a graph rewriting step. This intuitive description can be further specified in the context of the DPO approach, as illustrated in Figure 2.9.

- The part to be rewritten L constitutes the left-hand side of the rule.
- The rewritten part R constitutes its right-hand side.
- Their shared structure in blue constitutes the rule's interface K.

This interface allows reconnecting the rule to the graph. The part of the graph G to be modified is identified by the morphism $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$. The graph D is obtained by the removal of the green part, which is in L but not in K. Finally, the result of the rule H adds the orange content of R to D via K.

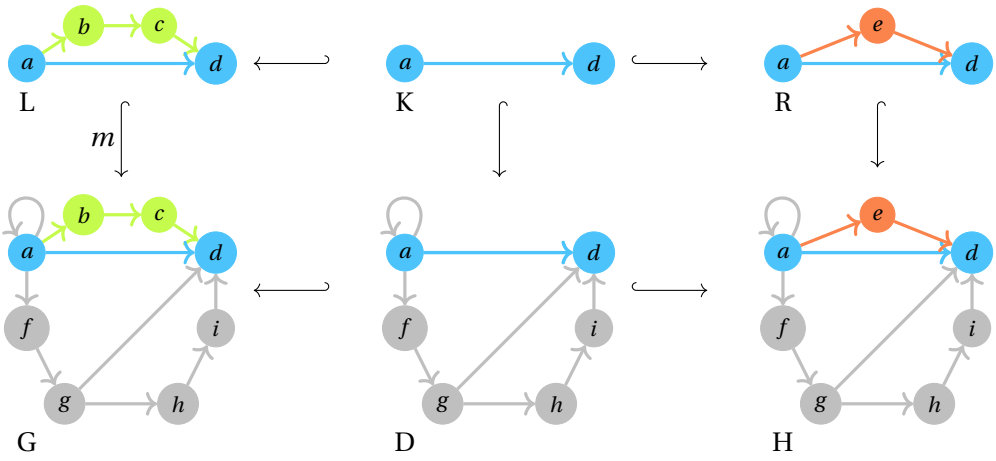


Figure 2.10: A direct derivation via the rules from Figure 2.8.

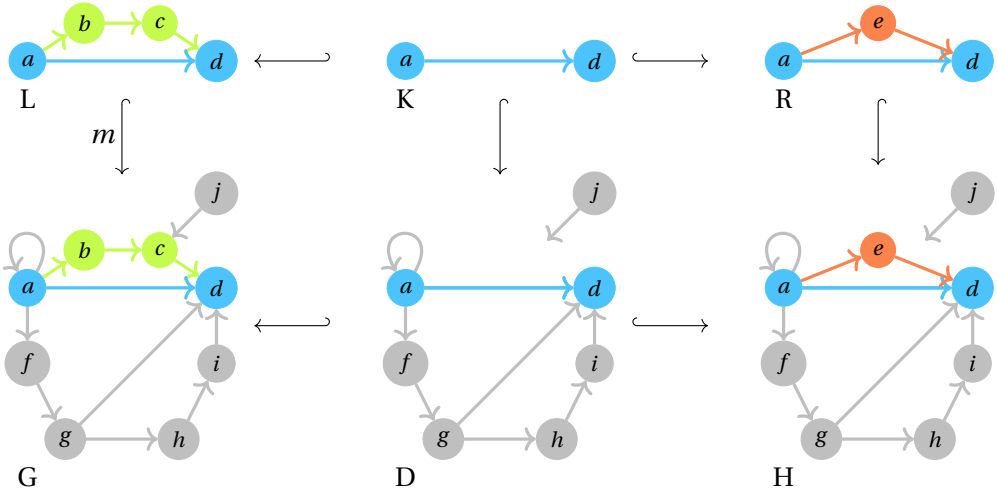


Figure 2.11: An invalid rewriting where the match does not satisfy the gluing condition.

Example 25 (Direct derivation)**.** Figure 2.10 presents an example of a direct derivation for the rule of Example 24. The direct derivation uses the same color code as in Figure 2.9: the interface is drawn in blue, the deleted part of the left-hand side in green, and the added part of the right-hand side in orange. Since the match admits a pushout complement, the direct derivation exists and yields the transformed graph H. On the contrary, the match of Figure 2.11 does not admit a pushout complement. Thus, the figure only gives an intuition of the modification. Node j is the source of an arc with no target in D and H. Thus, D and H are not graphs.

In a general category, the pushout complement of $K \to L \to G$ is not characterized by a universal property. However, in an adhesive category, if the pushout complement of monos exists, it is unique (up to isomorphism), ensuring the determinism of the rewriting. We could use rules where only $K \hookrightarrow L$ is a mono. However, we choose to stick to spans of monos, also called *linear rules*, for the ability to identify images and preimages of elements, which will prove fruitful for the various proofs. Linear rules are also the de facto standard notion used when studying rewriting in adhesive categories.

2.3.3 Restrictions and simplifications of the DPO approach

When dealing with modeling operations expressed as DPO rules, we will restrict DPO rewriting to its most conservative approach (see the summary "From Conservative to Radical" in [99]) and restrict matches to monomorphisms.

Assumption 1. *Matches are monomorphisms.*

Monic matches do not lessen the expressiveness of the rewriting system [88] but allow for explicit counting when deleting nodes or edges. Such matches also reduce the gluing condition to the dangling condition [60]: no node of $m(\mathrm{L}) \setminus m(\mathrm{K})$ is source or target of an arc in $\mathrm{G} \setminus m(\mathrm{L})$. We call *occurrence* of a match $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ the structure $m(\mathrm{L})$ conceived as the image of the node and arc functions defining m.

Together with the hypothesis of linear rules, this assumption means we consider double-pushout diagrams where all morphisms are monos. Note that the rule from Example 24 and its application in Example 25 satisfy this hypothesis.

When a rule $r = \mathrm{L} \overset{i_\mathrm{L}}{\hookleftarrow} \mathrm{K} \overset{i_\mathrm{R}}{\hookrightarrow} \mathrm{R}$ is linear, it can be thought of as the inclusion in R of a subgraph K of L, i.e., as a partial monomorphism $r = \mathrm{L} \rightharpoonup \mathrm{R}$. Adopting a functional, set-based point of view, we can describe r as follows [99].

- Some elements of L are mapped to some elements of R. These elements are said to be *preserved*. These correspond to the elements of K.
- Some elements of L are not mapped to some elements of R. These elements are said to be *deleted*. These correspond to the elements of $\mathrm{L} \setminus i_\mathrm{L}(\mathrm{K})$.
- Some elements of R are not mapped from some elements of L. These elements are said to be *added*. These correspond to the elements of $\mathrm{R} \setminus i_\mathrm{R}(\mathrm{K})$.

The notations $\mathrm{L} \setminus i_\mathrm{L}(\mathrm{K})$ and $\mathrm{R} \setminus i_\mathrm{R}(\mathrm{K})$ have to be read componentwise as set difference. Note that such constructions do not belong to category theory as $\mathrm{L} \setminus i_\mathrm{L}(\mathrm{K})$ and $\mathrm{R} \setminus i_\mathrm{R}(\mathrm{K})$ might not be graphs. However, our ambition to develop a formalization of modeling operations that can effectively be implemented sometimes leads us not to embrace the full theory of graph transformations (and thus the subjacent categorical framework).

In this sense, the interface K can be seen as the intersection of L and R. Like the set difference, this intersection is to be understood componentwise on the set of nodes and the set of arcs. This construction is also closely related to our actual implementation of rules (discussed thoroughly in Chapter 6). We represent rules by specifying a left-hand side L and a right-hand side R. We then injectively map some elements of L to some elements of R. This map corresponds exactly to a partial monomorphism of graphs. Under the hypothesis of linear rule, the interface corresponds to the elements of L that have an image in R or equivalently to the elements of R that have a preimage in L. The right mono $\mathrm{K} \overset{i_\mathrm{R}}{\hookrightarrow} \mathrm{R}$ then coincides with the map restricted to the interface K. The left mono $\mathrm{K} \overset{i_\mathrm{L}}{\hookrightarrow} \mathrm{L}$ associates each element of the restriction K to itself in L. Besides, we believe such an intuition might clarify rules for readers less familiar with graph transformations.

Notation 4.

1. *For a linear rules $r = \mathrm{L} \rightharpoonup \mathrm{R}$, we write $\mathrm{L} \cap \mathrm{R}$ for its interface, i_L for its left mono, and i_R for its right mono. In full, the rule is written $\mathrm{L} \overset{i_\mathrm{L}}{\hookleftarrow} \mathrm{L} \cap \mathrm{R} \overset{i_\mathrm{R}}{\hookrightarrow} \mathrm{R}$.*

2. *To alleviate the notations, we will omit the monos* i_L *and* i_R*, i.e. from structure* X *in* $L \cap R$*, we will call* X *the subobject* $i_L(X)$ *of* L *isomorphic to* X*, respectively the subobject* $i_R(X)$ *of* R *isomorphic to* X*. Intuitively, we are considering monos to be componentwise inclusions.*

3. *By extrapolation, we write* $X \in L \setminus R$ *for a deleted structure* X *in* $L \setminus i_L(L \cap R)$ *and* $X \in R \setminus L$ *for an added structure* X *in* $R \setminus i_R(L \cap R)$*.*

4. *All these notations are extended to spans of monos. Thus, we will also use them for the span* $G \hookleftarrow D \hookrightarrow H$ *of a direct derivation* $G \Rightarrow^{r,m} H$ *where* $r = L \rightarrowtail R$ *is a linear rule and* $m\colon L \hookrightarrow G$ *is a monic match.*

2.4 Applications of graph transformations

This section discusses some applications of graph transformations, emphasizing the ones in computer graphics and geometric modeling.

General applications and tools Formalizations based on graph transformations have been successfully achieved in many areas across computer science, such as database design [86, 96], concurrent and distributed systems [61], software engineering [99], IT landscape modeling [94]. Essentially, graph transformations provide a safe framework for studying the preservation of domain-related properties. We will use graph transformations to formalize geometric modeling operations in our case.

Extensions made to graph transformations ease their integration in generic tools such as PROGRES, AGG, Groove, or GrGen. PROGRES [160] enables 'graph grammar engineering,' which allows software prototyping based on graph rewriting. Groove [152] is a model checker for object-oriented systems based on graph transformations. AGG [171, 172] developed at TU Berlin is probably the most mature tool exploiting graph transformations with applications to specification and prototyping, although it can also be used as a graph transformation engine. In these tools, the more challenging problem is usually finding a match since it relies on finding an isomorphic subgraph, which is NP-complete. In AGG, the computation of subgraph isomorphism is solved as a constraint satisfaction problem, while GrGen [75] uses a notion called 'search plans' that corresponds to heuristic strategies. These strategies have a cost, meaning that finding the right search plan becomes an optimization problem that depends on the host graph.

Graph transformations operate at a low level on graph structures, meaning that rules are self-contained. Therefore, a single rule application engine tailored to a given graph transformation class enables the application of all domain-related rules. These engines justify the development of dedicated tools, such as Fujaba [133] for code refactoring, Gremlin [155] for queries in graph databases, DiaGen [130] for the manipulation of diagrams, GReTL [52] for the parallel construction of metamodels and conforming models, or Grape [181] and it most recent extension GrapeVine [182] for the modification of persistent graphs, i.e., graphs with history. This point is crucial in designing a generic modeling tool as it minimizes the implementation and debugging efforts.

Graph rewriting in computer graphics Within computer graphics, formalizations have also been accomplished with graph transformations. For instance, in [188], attributed grammars were defined to describe and classify the syntactic structures of two-dimensional shapes. More recently,

indoor scenes have been represented as graphs in [105], where a graph grammar allowed the reconstruction of scenes from images acquired by cameras. In [177], graphs describe the modeling process of tunnels, and graph transformations specify modifications to this process. The main idea is to handle consistency between different levels of detail in the infrastructure during transformations. In [135], multiscale modeling of plants also takes advantage of a graph grammar called the Relational Growth Grammar. In [69], a geometric graph grammar is presented based on a node labeled controlled (NLC) graph grammar. NLC graph grammars allow rewriting star graphs (i.e., a node plus its incident arcs) by a more general graph, where the reconnection is ensured by the other extremities of the incident edges. These geometric graph grammars encode a city's geometry and topology via its road network. They can be used for generating cities by standard application but also enable learning the grammar by finding isomorphic subgraphs within the network.

Previous works introduced a graph-based representation of generalized maps for geometric modeling [145] and used DPO graph transformations to design modeling operations [144]. The representation of objects also relies on other data, such as vertex positions, edge curvatures, or volume densities. In topology-based geometric modeling, these data are referred to as embedding. In [15], embeddings were defined as a family of node labels with appropriate consistency constraints and conditions. Since rules can modify both the topology and the embeddings of an object, conditions to preserve the embedding consistency in meta-rules were studied in [14]. In [145, 16], DPO rewriting was enriched with variables to construct meta-rules instantiated with a set-based operation similar to a pullback. In [144], sufficient consistency preservation conditions were introduced for DPO transformations, and only the incident arcs condition was lifted to meta-rules. Based on these works, a platform called Jerboa [12] has been developed to prototype dedicated geometric modelers.

Two other lines of work can be found in the literature where similar applications can be derived. The programming language MGS [76] originates from the simulation of biological processes and dynamical systems with the idea that the topology of collections matters when performing computations. Therefore, MGC allows the manipulation of various data structures called topological collections. Among the various topological collections, MGS enables the use of abstract cellular complexes, hence offering a language to design modeling operations. A construction of Sierpinski triangles and Menger sponges is presented in [166], while the Loop subdivision algorithm is presented in [167].

More recently, [27] showed that specifications from category theory can be implemented efficiently with imperative programming. The underlying structure is $\mathscr{C}$-sets which correspond to a copresheaf where $\mathscr{C}$ is viewed as a category, i.e., a functor $\mathscr{C} \to \mathbf{Set}$, which correspond to a generalization of graphs. The authors explain how to represent simplicial complexes (a generalization of the triangulation of a surface to higher dimensions) using $\mathscr{C}$-sets, thus enabling the representation of geometric objects.

Chapter 3

Topological rewriting of combinatorial maps

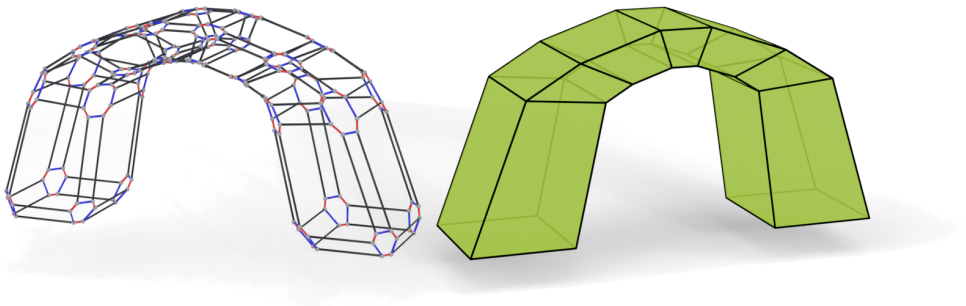


Figure 3.1: The arch-like object on the right can be represented as a 2-Gmap, i.e., as the graph displayed on the left.

Personal note on the chapter

In this chapter, I provide an abstract representation of generalized and oriented maps as graphs labeled by dimensions and subject to structural constraints. The presentation relies on the algebraic approach to graph transformations to provide a formalization of geometric modeling operations.

To the reader familiar with computer graphics and geometric modeling, the proposed level of abstraction might appear too heavy, if not straight-out scary. A simpler presentation can be conceived with functions on sets. This point of view is detailed in the second part of this dissertation, without resorting to category theory. However, I wish to say that my approach to the inference of modeling operations is heavily rooted in this categorical framework. Therefore, I believe that the formalization of modeling operations can be a real asset for reasoning about these operations.

To the reader familiar with category theory and graph rewriting, the proposed level of abstraction might seem lackluster compared to more recent developments within the community. Nonetheless, most choices made come from needs related to the application domain. In particular, our ambition is to provide a tool for the design of modeling operations, i.e., to help the first kind of readers write graph transformation rules without needing to understand the underlying theory. Therefore, we strive to design an accessible framework for less category-aware users.

I humbly invite any reader to question themselves about the sense and use to give to abstraction. Abstraction comes from the Latin word 'abstraho', which means 'pull away'. Abstraction is about removing details. These details allow grasping the differences between otherwise similar things. Thus, abstracting means that even though things may differ, it does not necessarily matter, and we can consider them to be the same. The key idea is to apply the same treatment to things that behave similarly. Only then can more profound reasoning be understood. However, too much abstraction may have the undesired effect that we can no longer see the ideas needed to solve a given problem. Given their goal, one should ask what the proper level of abstraction is and accept that other issues might require more, or less, abstraction.

Contents

Generalized and oriented maps admit a combinatorial definition (see Chapter 1). In this chapter, we adapt the combinatorial point of view to build a graph-based framework. We call Gmaps and Omaps the respective graph counterparts of generalized and oriented maps. Generalized and oriented maps belong to the family of combinatorial maps, providing a combinatorial representation of space subdivisions in arbitrary dimensions. Such representations rely on permutations on a set of darts to describe the relations between the topological cells of a subdivision describing an object. Each permutation can be transformed into a graph, meaning that a combinatorial map can be thought of as a graph labeled on its arcs by the permutation name. However, any graph labeled on the related alphabet does not yield a valid combinatorial map. Indeed, we have to ensure that each label describes a permutation. The permutations are also subject to some involution properties for the data to be a well-formed combinatorial map. Besides, these properties vary based on the specific model, e.g., Gmap, Omap. Therefore, we need to add more information to ensure that a given labeled graph is indeed a combinatorial map.

Typed graphs (or even attributed graphs used later in Chapter 5) do not allow the encoding of the *consistency* of combinatorial maps into a graph. The standard approach in the graph transformation community is to define *consistency constraints*, or *constraints* for short, on the graphs. Constraints describe properties on graphs and, more generally, on objects, which must be satisfied. For instance, constraints can describe the specification of a dynamic system. Indeed, graph transformations have been heavily used to model dynamic systems where one graph represents a possible state of the system, and a derivation represents an evolution of the system. In such cases, knowing which states can be reached and which cannot is often crucial. In particular, some states may violate the system specification, especially for a nontrivial specification. Thus, the system specification partitions the state space into consistent states that fulfill the specification and inconsistent states that violate them.

Starting from a consistent graph, i.e., a graph that satisfies the considered constraints, the question is whether applying a rule yields a still consistent graph. When the goal is the study of consistency preservation, constraints are considered to be invariants over graphs. Ensuring that a constraint holds after a transformation is called *preservation* [99, Chap. 4] if it held before the transformation and *guarantee* if we do not know (or do not care) whether it held before. In this dissertation, we are motivated by the preservation of constraints to ensure the well-formedness of combinatorial models through modeling operations. If this can be proven for all rules and all consistent graphs, then all reachable graphs are consistent.

One could consider the full subcategory of consistent objects to obtain constraint-preserving transformations. This full subcategory consists of all consistent objects and all morphisms between these objects. Such a restriction may result in a loss of expressiveness or at least make writing some rules artificially tricky. Therefore, it is often preferable to work with objects that satisfy the constraints but with rules written with objects that do not satisfy them. In other words, it is often suitable to work within a larger category, which raises the question of whether a given graph satisfies the constraints. If the constraints are easy to check, one could perform a routine verification after transforming the graph. However, checking the constraints can be challenging in some cases, e.g., for large graphs, where one would need to apply the transformation, check the constraints and then revert the transformation if the constraints are not satisfied. Therefore, a more fruitful approach is to study conditions at the rule level that guarantee, or preserve, given con-

straints on the resulting graph. From a more operational point of view, conditions offer a control mechanism over the system by restricting the applicability to transformations that yield consistent graphs.

Conditions on rules for consistency preservation were provided in [144] for the specific case of generalized maps. These conditions led to the definition of a generic topological-based geometric modeler in [16] based on Gmaps. Geometric modeling operations can be conceived and written as simple rules. In this chapter, we extend previous works in two ways. First, we use a local characterization of the constraints which means that the study encompasses more models belonging to combinatorial maps. The results presented here allow dealing with both oriented and generalized maps. Secondly, we derive more precise conditions for rules. More precisely, we derive necessary and sufficient conditions while previous works only provided sufficient ones, leading to false negatives when verifying the rules.

In this chapter, we first present some definitions and notations related to labeled graphs in Section 3.1. In Section 3.2, we give a graph-based definition of the combinatorial models via the study of three topological constraints. Then, we discuss the preservation of the model consistency in Section 3.3 and establish conditions on rules related to the topological constraints. Our constraints are written in a set-based framework that supports necessary and sufficient conditions for consistency preservation. In particular, our conditions can be efficiently checked simultaneously as a user writes a rule. Indeed, the main motivation of the formal framework, presented in the first part of this dissertation, is to assist the user in designing sound operations. Therefore, we strive to obtain conditions that can be efficiently checked at the same time as a user is writing a rule, i.e., at design time. Motivated by efficient verification, our approach for consistency preservation differs from current methods in the graph rewriting community. These methods are reviewed and compared to our approach in Section 3.4. The main contributions of this chapter, namely Sections 3.2 and 3.3, have been published in [139].

3.1 Arc-labeled graphs

In Section 3.2, we will define the combinatorial models of generalized and oriented maps as arc-labeled graphs subject to some constraints. In this section, we discuss the labeling part of the construction. Labels are decorations added to the graphs, i.e., information added to the relations described by the graph's structure. The set of labels forms an alphabet. Thus, we first recall some definitions and notations related to alphabets, letters, and words.

Definition 26 (Alphabet)**.** *An* alphabet Σ *is a set of atomic elements called* letters. *A* word *on* Σ *is a finite sequence of letters from* Σ. *The* empty word ϵ *is the word with no letter. The set of all words on* Σ *is written* Σ^*. *A word* w *on an alphabet* Σ *is of* length n *in* $\mathbb{N}$, *denoted by* $|w| = n$, *if it is a sequence of* n *letters. The* k*-th* ($0 < k \leq |w|$) *letter of a word* w *is written* $w_{(k)}$. *The* concatenation *of two words* u *and* v *from* Σ^* *is the word* uv *of length* $|u| + |v|$ *such that* $(uv)_{(k)} = u_{(k)}$ *for all* $0 < k \leq |u|$, *and* $(uv)_{(|u|+k)} = v_{(k)}$ *for all* $0 < k \leq |v|$.

The addition of labels to a graph was first described by adding a labeling function to the definition of a graph, see [55, 93, 99]. In some papers, labels are also called colors. Intuitively, it suffices to extend the diagrammatic definition of graphs (the purple part corresponds to the diagram 2.2

in Chapter 2) to add labeling alphabets and functions for both nodes and arcs, (Σ_V and l_V for the nodes, Σ_E and l_E for the arcs):

$$\Sigma_E \xleftarrow{le} E \underset{t}{\overset{s}{\rightrightarrows}} V \xrightarrow{lv} \Sigma_V$$

Definition 27 (Labeled graph and morphism)**.** *Consider a pair of alphabets* (Σ_V, Σ_E)*. A* graph labeled over (Σ_V, Σ_E) *is a tuple* $G = (V_G, E_G, s_G, t_G, lv_G, le_G)$ *where* (V_G, E_G, s_G, t_G) *forms a graph while* $lv_G : V_G \to \Sigma_V$ *and* $le_G : E_G \to \Sigma_E$ *are respectively the node and arc* label functions*. Labeled-graph morphisms preserve labels, meaning that for a morphism* $m = (m_V, m_E) : G \to H$*, the following triangles commute in* **Set**.

$$\begin{array}{ccc} V_G & \xrightarrow{m_V} & V_H \\ & {\scriptstyle lv_G}\searrow \quad \swarrow{\scriptstyle lv_H} & \\ & \Sigma_V & \end{array} \qquad \begin{array}{ccc} E_G & \xrightarrow{m_E} & E_H \\ & {\scriptstyle le_G}\searrow \quad \swarrow{\scriptstyle le_H} & \\ & \Sigma_E & \end{array}$$

Labeled graphs and their morphisms form a category, which can be seen as a special case of typed graph category (Definition 19 in Chapter 2).

Example 26 (Labeled graphs [57])**.** Let Σ_V and Σ_E respectively be labeling alphabets for the set of nodes and arcs. The category of graphs labeled over (Σ_V, Σ_E) is equivalent to the category $\mathbf{Graph}_{\mathbf{TG}(\Sigma_V, \Sigma_E)}$ of graphs typed over the graph $TG(\Sigma_V, \Sigma_E) = (V_{(\Sigma_V,\Sigma_E)}, E_{(\Sigma_V,\Sigma_E)}, s_{(\Sigma_V,\Sigma_E)}, t_{(\Sigma_V,\Sigma_E)})$ such that:

- $V_{(\Sigma_V,\Sigma_E)} = \Sigma_V$;
- $E_{(\Sigma_V,\Sigma_E)} = \Sigma_E \times \Sigma_V \times \Sigma_V$;
- $s_{(\Sigma_V,\Sigma_E)} : E_{(\Sigma_V,\Sigma_E)} \to V_{(\Sigma_V,\Sigma_E)};\ (e, a, b) \mapsto a$;
- $t_{(\Sigma_V,\Sigma_E)} : E_{(\Sigma_V,\Sigma_E)} \to V_{(\Sigma_V,\Sigma_E)};\ (e, a, b) \mapsto b$.

Note that commutative triangles of Definition 27 are similar to the commutative triangle of a slice category (Definition 7) defining typed graph. The type-based definition of labeled graphs ensures that labeled graphs form an adhesive category (Definition 21), meaning that DPO rewriting is well defined.

To specify our combinatorial models, we only require labels over the set of arcs. Therefore, we introduce some notations about the category of arc-labeled graphs over an alphabet Σ.

Notation 5.

1. *Given a finite alphabet* Σ*, we write* Σ-**Graph** *for the category of* graphs arc-labeled *on* Σ*, i.e., the category* $\mathbf{Graph}_{\mathbf{TG}(\emptyset,\Sigma)}$ *following the notations from Example 26.*
2. *Like any object of* Σ-**Graph**, *the terminal object* $\mathbb{1}_\Sigma$ *consists of a graph and a morphism. The graph is the graph with one node and one loop per letter in* Σ*, while the morphism is the identity associated with this graph. We will also write* $\mathbb{1}_\Sigma$ *to denote the graph with one node and one loop per letter in* Σ*, i.e., without accounting for the associated morphism.*

3. *We write* l_G *for the morphism* $G \to 1_\Sigma$ *in* **Graph**, *meaning that an object in* Σ-**Graph** *is a pair* (G, l_G). *We might directly refer to the graph* G *as the object in* Σ-**Graph** *and write it as* $G = (V_G, E_G, s_G, t_G, l_G)$. *We write* $l_G(e)$ *or simply* $l(e)$ *for the label of an arc* e *in* (G, l_G).

4. *We identify a morphism* $m = (m_V, m_E) : G \to H$ *with its component-wise functions and omit the subscripts* V *and* E*: we write* $m(x)$ *regardless of whether* x *is a node or an arc of* G.

5. *We identify arcs in the graph using their labels: an arc* e *labeled by* i *in* Σ*, i.e., such that* $l(e) = i$ *is an* i-arc.

6. *We will consider products in various categories* Σ-**Graph**, *i.e., with different alphabets* Σ. *Therefore, we might subscript the product by the alphabet for clarity:* $A \times_\Sigma B$ *means that the product lives in* Σ-**Graph**.

In the category Σ-**Graph**, a path $e_1 \dots e_n$ is a w-*path* if e_k is a $w_{(k)}$-arc for every $1 \leq k \leq n$. If the path is a cycle, then it is a w-cycle. A w-path $p = e_1 \dots e_n$ can be denoted by $s(e_1) \overset{w}{\rightsquigarrow} t(e_n)$ when we are only interested in the source, target and label of the path.

In the combinatorial models, the labels allow the encoding of dimensional relations between the object subparts. In this chapter we use categories $\mathbb{D}$-**Graph** where $\mathbb{D}$ is a finite set of integers, e.g, $\mathbb{D} = 1..4$, the set of integers between 1 and 4 (included). As integers describe the neighboring relations in topological models, we call topological any graph labeled with integers from $\mathbb{D}$. An example of topological graph is presented in Figure 3.2.

Definition 28 (Topological graph)**.** *Let* $\mathbb{D}$ *be a finite set of integers, called* dimension set. *A* $\mathbb{D}$-topological graph *is a graph from the category* $\mathbb{D}$-**Graph**.

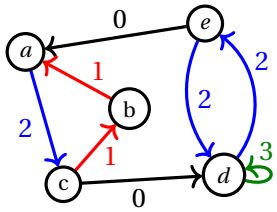


Figure 3.2: A (topological) (0..3)-graph.

Example 27 (Topological graph)**.** The graph G of Figure 3.2 is a topological (1..3)-graph. We will use similar graphs in this chapter with different labeling sets, all intended to represent a set of dimensions. The graph G also contains the complete color code used in this dissertation: black (—) for the dimension 0, red (—) for the dimension 1, blue (—) for the dimension 2, and green (—) for the dimension 3. Dimensions are often omitted in the figures and represented by the arcs' color.

3.2 Generalized and oriented maps

As seen in Chapter 1, generalized and oriented maps admit a combinatorial definition with constrained involutions over a set of darts. Poudret et al. introduced a graph-based definition of generalized maps in [144]. Here, we extend their work by designing a framework encompassing generalized maps (Gmaps) and oriented maps (Omaps) by considering the key constraints separately.

3.2.1 Topological constraints

The combinatorial definition of oriented maps and generalized maps [43] exploits a set of permutations $P_1, \ldots, P_n$ on a set of darts D. We consider each permutation P_i as a relation over D, i.e., a subset of $D \times D$. Therefore, we can translate the structure $< D, P_1, \ldots, P_n >$ into a graph.Each dart is considered as a node of the graph, while each permutation P_i yields the set of i-labeled arcs linking these nodes. The final set of edges consists of the union of the P_i's. An illustration of this construction is provided in Figure 3.3. Both graphs 3.3a and 3.3b share $D = \{a, b, c, d, e\}$ for the set of nodes and have a set of arcs deduced from a permutation, respectively drawn in red and blue. The final graph 3.3c has the same nodes and the union of the arcs from the two graphs. Since the combinatorial definition [43] of Gmaps sub-scripts permutations by the adequate dimension, we label each arc with the suitable dimension.

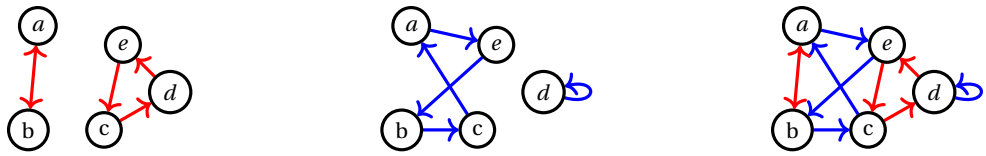


Figure 3.3: Representation of the permutations $\{P_1, P_2\}$ over the set $D = \{a, b, c, d, e\}$ as a graph: (a) $P_1 = \{(a, b), (b, a), (c, d), (d, e), (e, c)\}$ represented as a graph G_1; (b) $P_2 = \{(a, e), (b, c), (c, a), (d, d), (e, b)\}$ represented as a graph G_2; (c) $< D, P_1, P_2 >$ represented as a graph $G_{1,2}$ whose edge set is $E_{G_1} \cup E_{G_2}$. Similar examples can be found in [43, Chap. 2.5.2].

Any graph does not describe a permutation. Moreover, the set of permutations defining a combinatorial map, i.e., a generalized map or an oriented map, is subject to some constraints; namely, some (compositions of) permutations are involutions. Following the construction of [144], we introduce properties called *topological constraints* that a graph should satisfy to be a generalized map, respectively, an oriented map. We propose a local, dimension-based definition of these properties.

Incident arcs constraint

The first property ensures that each label induces the graph of a permutation. In categorical words, a permutation is an isomorphism from an object to itself. In combinatorics, a permutation is a bijection from a set to itself. Therefore, a permutation admits a local characterization: each element in the set must admit an image by the function and a unique preimage. In the graph jargon, each node should be the source and target of a unique arc. Given a node v, the arcs that have e as source or target are called incident to e. Thus, we call this property the property of incident arcs.

Definition 29 (Incident arcs constraint)**.** *Let* $G = (V_G, E_G, s_G, t_G, l_G)$ *be a* Σ*-labeled graph,* v *be a node of* V_G*, and* i *a labels of* Σ*. The node* v *satisfies the* incident arcs constraint $\mathtt{I}_G(i)$ *if* v *is the source of a unique* i*-arc and the target of a unique* i*-arc. We write* $v \models \mathtt{I}_G(i)$ *if the node* v *satisfies* $\mathtt{I}_G(i)$ *and* $v \not\models \mathtt{I}_G(i)$ *otherwise.*

The constraint can be extended to a subset of labels $I \subseteq \Sigma$ *and a subset of nodes* $V' \subseteq V_G$*. We say* G *satisfies the constraint* $\mathtt{I}_{V',G}(I)$*, written* $G \models \mathtt{I}_{V',G}(I)$*, whenever each node in* V' *satisfies the constraint for all labels in* I*. If* $V' = V_G$*, we simply say* G *satisfies* $\mathtt{I}_G(i)$ *or* $\mathtt{I}_G(I)$*.*

The incident arcs constraint can be written in logical terms following Courcelle' formalism [37, 38]. We consider quantifications over nodes and arcs, a ternary incidence relation $\mathbf{i}$, a binary equality relation $=$, and add a unary predicate $\mathbf{l}_i$ for each label in Σ. The intuition is that $\mathbf{i}(e,u,v)$ expresses that e is an arc from u to v, while $\mathbf{l}_i(x)$ means that x is labeled by i. These predicates induce a signature that inductively defines a set of graph formulas via the usual first-order quantifiers and connectives. Following the abbreviation of [91], we write $\mathbf{n}(x)$ for $\neg\exists y\exists z, \mathbf{i}(x,y,z)$, intuitively meaning that the variable x represents a node. For a non-empty graph $G = (V_G, E_G, s_G, t_G, l_G)$, the semantics of a formula in G is given by the interpretation ν_G such that:

- $\nu_G(\mathbf{i}(e,u,v)) = 1$ if and only if $e \in E_G$, $u \in V_G$, $v \in V_G$, $s_G(e) = u$, $t_G(e) = v$,
- $\nu_G(x = y) = 1$ if and only if $x = y$,
- $\nu_G(\mathbf{l}_i(x) = 1)$ if and only if $l_G(x) = i$.

The incident arcs constraint can then be rewritten with constraints $\mathtt{I}_1$ to $\mathtt{I}_4$. Constraints $\mathtt{I}_1$ and $\mathtt{I}_3$ state that an arc of source, respectively target, v exists in the graph. Constraints $\mathtt{I}_2$ and $\mathtt{I}_4$ state that these arcs are unique, with the habitual trick stating that if two such arcs exist, there are equal.

$$\mathbf{n}(v) \implies (\exists e \exists u\ \mathbf{i}(e,v,u) \wedge \mathbf{l}_i(e)) \tag{$\mathtt{I}_1$}$$

$$\mathbf{n}(v) \implies (\forall e \forall e' \forall u \forall u'\ \mathbf{i}(e,v,u) \wedge \mathbf{l}_i(e) \wedge \mathbf{i}(e',v,u') \wedge \mathbf{l}_i(e') \implies e = e') \tag{$\mathtt{I}_2$}$$

$$\mathbf{n}(v) \implies (\exists e \exists u\ \mathbf{i}(e,u,v) \wedge \mathbf{l}_i(e)) \tag{$\mathtt{I}_3$}$$

$$\mathbf{n}(v) \implies (\forall e \forall e' \forall u \forall u'\ \mathbf{i}(e,u,v) \wedge \mathbf{l}_i(e) \wedge \mathbf{i}(e',u',v) \wedge \mathbf{l}_i(e') \implies e = e') \tag{$\mathtt{I}_4$}$$

In other words, for a node v in V_G, we have $v \models \mathtt{I}_G(i)$ if and only if the constraints $\mathtt{I}_1$ to $\mathtt{I}_4$ hold. Besides, we have the following equivalences, which essentially reformulate closure over the considered sets.

- $v \models \mathtt{I}_G(I) \equiv \forall i \in I, v \models \mathtt{I}_G(i)$
- $G \models \mathtt{I}_{V',G}(i) \equiv \forall v \in V', v \models \mathtt{I}_G(i)$
- $G \models \mathtt{I}_{V',G}(I) \equiv \forall v \in V', v \models \mathtt{I}_G(I) \equiv \forall i \in I, G \models \mathtt{I}_{V',G}(i)$
- $G \models \mathtt{I}_G(i) \equiv G \models \mathtt{I}_{V_G,G}(i)$
- $G \models \mathtt{I}_G(I) \equiv G \models \mathtt{I}_{V_G,G}(I)$

Example 28 (Incident arcs constraint)**.** The graph $G_{1,2}$ of Figure 3.3c represents two permutations. Thus, we have $G_{1,2} \models \mathtt{I}_{G_{1,2}}(\{1,2\})$.

Non-orientation constraint

Since the combinatorial models impose that some permutations are involutions, we introduce a second property, ensuring that a label in the graph describes a permutation corresponding to its own inverse. Therefore, an involution is a symmetric relation over a set. Intuitively, the induced

graph should be undirected. Since all permutations need not be involutions, the constraint is defined on a per-label basis rather than directly working in the category of undirected graphs. Thus, we enforce that arcs should be non-oriented, once again in a local manner.

Definition 30 (Non-orientation constraint)**.** *Let* $G = (V_G, E_G, s_G, t_G, l_G)$ *be a* Σ*-labeled graph,* v *be a node of* V_G*, and* i *a labels of* Σ*. The node* v *satisfies the* non-orientation constraint $0_G(i)$ *if the* i*-arcs incident to* v *are non-oriented. We write* $v \models 0_G(i)$ *if the node* v *satisfies* $0_G(i)$ *and* $v \not\models 0_G(i)$ *otherwise.*

The constraint can be extended to a subset of labels $I \subseteq \Sigma$ *and a subset of nodes* $V' \subseteq V_G$*. We say* G *satisfies the constraint* $0_{V',G}(I)$*, written* $G \models 0_{V',G}(I)$*, whenever each node in* V' *satisfies the constraint for all labels in* I*. If* $V' = V_G$*, we simply say* G *satisfies* $0_G(i)$ *or* $0_G(I)$*.*

Once again, the non-orientation constraint can be narrowed to logic formulas. In this case, there are only two, one for the arcs of source v and one for the arc of target v.

$$\mathbf{n}(v) \implies (\exists e \exists u\ \mathbf{i}(e, v, u) \wedge l_i(e) \implies \exists e'\ \mathbf{i}(e', u, v) \wedge l_i(e')) \qquad (0_1)$$

$$\mathbf{n}(v) \implies (\exists e \exists u\ \mathbf{i}(e, u, v) \wedge l_i(e) \implies \exists e'\ \mathbf{i}(e', v, u) \wedge l_i(e')) \qquad (0_2)$$

All the equivalences presented for the incident arcs constraint also hold with the non-orientation one. A watchful reader will have remarked that the non-orientation property alone does not ensure that the function represented by the graph is an involution.

Graphically, we will draw non-oriented arcs either as two-sided arrows to emphasize that some arcs may be oriented or as (curved) lines without arrows when the graph is undirected, i.e., all arcs are non-oriented.

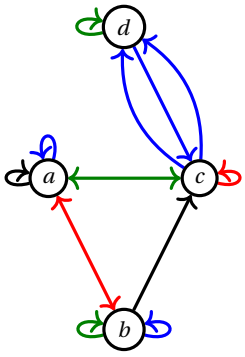


Figure 3.4: A topological (0..3)-graph G satisfying $0_G(1..3)$.

Example 29 (Non-orientation constraint)**.** The graph G of Figure 3.4 satisfies the non-orientation constraint for the dimension 1, 2, and 3, but not 0.

- The 0-arc between nodes b and c does not admit any reverse arc, thus nodes b and c do not satisfy $0_G(0)$, thus $G \not\models 0_G(0)$.
- All 1-arcs admit a reverse arc, meaning $G \models 0_G(1)$. Note that the 1-loop on c is its own reverse arc. However, the function associated with the 1-arcs is not an involution. Indeed, node d would have no image by such a function (and no preimage).
- Likewise, all 2-arcs admit a reverse arc and $G \models 0_G(2)$. However, the relation associated

with the 2-arcs is not a function since two 2-arcs go from c to d: the 2-arcs from d to d admits a reverse arc, but it is not unique. Therefore, the label 2 does not represent an involution.

- Finally all 3-arcs admit a reverse arc and $G \models \mathtt{O}_G(3)$. Besides, all reverse arcs are unique, and the function associated with 3 models an involution.

Example 29 emphasizes that the constraints should be considered hierarchically. It is easier to consider that relations are involutions if they are permutations. Therefore, we will mainly consider the non-orientation constraint through the prism of the incident arcs constraint.

Cycle constraint

The last property used in the definition of combinatorial maps states that some compositions of permutations should be involutions. Intuitively, this property means that given a sequence of arc labels, if we follow it twice, we return to the starting point. In any graph, this property is ill-formed, and a node could not be the source of an arc for a given label (meaning we cannot follow the path) or could be the source of several arcs for a given label (meaning that the path may end on different nodes). Similar to the non-orientation constraint, designing a constraint for this property is heavily simplified if the incident arcs constraint holds. For our concern, we only require the composition of two permutations, meaning the sequence of labels is of length two, and we are looking for cycles of length four.

Definition 31 (Cycle constraint)**.** *Let* $G = (V_G, E_G, s_G, t_G, l_G)$ *be a* Σ*-labeled graph,* v *be a node of* V_G*, and* (i, j) *a pair of labels of* Σ*. The node* v *satisfies the* cycle constraint $\mathtt{C}_G(i, j)$ *if* v *is the source of an* $ijij$*-cycle. We write* $v \models \mathtt{C}_G(i, j)$ *if the node* v *satisfies* $\mathtt{C}_G(i, j)$ *and* $v \not\models \mathtt{C}_G(i, j)$ *otherwise.*

The constraint can be extended to a subset of pairs of labels $J \subseteq \Sigma^2$ *and a subset of nodes* $V' \subseteq V_G$*. We say* G *satisfies the constraint* $\mathtt{C}_{V',G}(J)$*, written* $G \models \mathtt{C}_{V',G}(J)$*, whenever each node in* V' *satisfies the constraint for all pairs of labels in* J*. If* $V' = V_G$*, we simply say* G *satisfies* $\mathtt{C}_G(i, j)$ *or* $\mathtt{C}_G(J)$*.*

The cycle constraint also admits a logical description that states the existence of a cycle starting from v (first line) where the arcs are adequately labeled (second line).

$$\mathbf{n}(v) \implies (\exists e_1 \exists e_2 \exists e_3 \exists e_4 \exists u_1 \exists u_2 \exists u_3 \;\; \mathbf{i}(e_1, v, u_1) \wedge \mathbf{i}(e_2, u_1, u_2) \wedge \mathbf{i}(e_3, u_2, u_3) \wedge \mathbf{i}(e_4, u_3, v) \\ \wedge l_i(e_1) \wedge l_j(e_2) \wedge l_i(e_3) \wedge l_j(e_4)) \tag{C}$$

When dealing with the cycle constraint, we call the set of *exchangeable dimensions* the set $J \subseteq \Sigma^2$ of pairs of labels. Note that the incident arcs constraint guarantees the existence and uniqueness of a path given a sequence of labels, which will massively simplify the study of consistency for the cycle constraint. Indeed, if a graph satisfies the incident arcs constraint for two dimensions, then the order of the dimensions does not matter when considering the cycle constraint at the whole graph scale.

Lemma 7. *Let* G *be a* $\mathbb{D}$*-topological graph and* i*,* j *two dimensions in* $\mathbb{D}$*. Suppose that* G *satisfies* $\mathtt{I}_G(\{i, j\})$*, then* G *satisfies* $\mathtt{C}_G(i, j)$ *if and only if* G *satisfies* $\mathtt{C}_G(j, i)$*.*

Proof. The lemma holds trivially from the incident arcs constraint (Definition 29). □

Before defining generalized and oriented maps, we recapitulate the three topological constraints. We use the same graph as in Figure 3.2, which is given again for readability purposes.

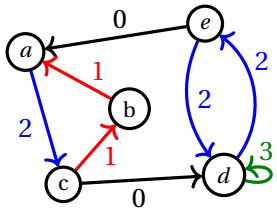


Figure 3.5: A (0..3)-graph.

Example 30 (Topological constraints)**.** Let G be the (0..3)-graph represented in Figure 3.5. Node b is the source of a single 1-arc and the target of a single 1-arc. Thus, node b satisfies the incident arcs constraint $\mathtt{I}_G(1)$. On the contrary, node c is not the source of a 2-arc and does satisfy $\mathtt{I}_G(2)$. Since there is a 2-arc of source d and target e and a 2-arc of source e and target d, nodes d and e satisfies the non-orientation constraint $\mathtt{O}_G(2)$. Similarly, since the only 3-arc incident to node d is a loop, it also satisfies $\mathtt{O}_G(3)$ and therefore satisfies $\mathtt{O}_G(\{2,3\})$. Conversely, node a does not satisfy $\mathtt{O}_G(2)$ because the 2-arc between nodes a and c does not admit a reverse arc. Note that no 2-arc is incident to node b, thus node b also satisfies $\mathtt{O}_G(2)$. Finally, as node c is the source of a 0202-cycle, node c satisfies the constraint $\mathtt{C}_G(0,2)$, whereas node a is the source of a 2020-cycle and satisfies $\mathtt{C}_G(2,0)$.

3.2.2 Graph-based definitions of combinatorial maps

The incident arcs constraint allows us to unambiguously consider arcs based on their label and source (or target). This feature will prove helpful in generalizing graph transformations. Therefore, we call combinatorial any topological graph that satisfies the incident arcs constraint on every dimension. We define Omaps and Gmaps based on combinatorial graphs.

Definition 32 (Combinatorial graphs, Gmaps, and Omaps)**.** *Let $\mathbb{D}$ be a finite subset of $\mathbb{N}$ and let $\mathbb{D}_{+2}$ denotes the set of pairs $(i,j) \in \mathbb{D}^2$ such that $i+2 \leq j$.*

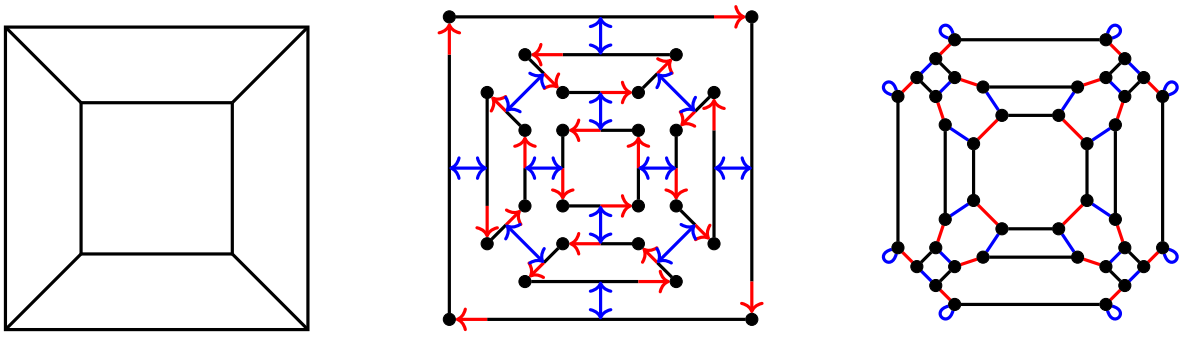
A $\mathbb{D}$-topological graph is a $\mathbb{D}$-combinatorial graph *if it satisfies the incident arcs constraint $\mathtt{I}_G(\mathbb{D})$.*

*An n-*Gmap *is a $(0..n)$-combinatorial graph satisfying $\mathtt{O}_G(0..n)$ and $\mathtt{C}_G((0..n)_{+2})$.*

*An n-*Omap *is a $(1..n)$-combinatorial graph satisfying $\mathtt{O}_G(2..n)$, and $\mathtt{C}_G((1..n)_{+2})$.*

By twisting the constraints, it is possible to define other models of [43]. For instance, a closed generalized map can be defined as a Gmap where no arc is a loop. An open-oriented map can be defined by relaxing the incident arcs condition on the maximal dimension n to "every node v is the source of, at most, one n-arc and the target of, at most, one n-arc". Similarly, the dual model of oriented maps (open or closed) can be obtained by enforcing the non-orientation condition on the minimal dimension 1 and removing it on the maximal dimension n.

We extend the study of [144] to cover Omaps, and one could extend it to the other models. We chose to keep the study simple by narrowing the scope to these two models as they are the more regular ones, simplifying the transformation mechanism's automatization.



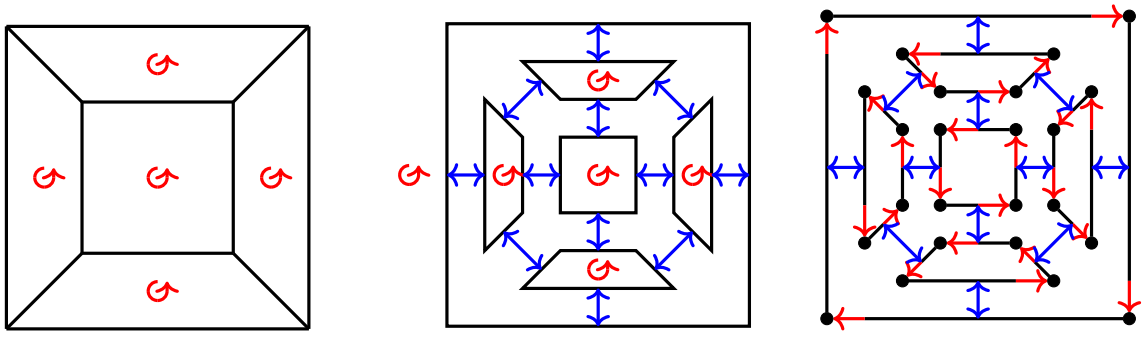
(a) (b) (c)

Figure 3.6: Comparison between the models: (a) an object, (b) its representation as a 2-Omap, and (c) as a 2-Gmap.

Example 31 (Gmap and Omap)**.** Let us illustrate both models with the example of Figure 3.6. In this figure, we draw the graphs closer to the usual representation in geometric modeling. In the 2-Omap (Fig. 3.6b), the darts are displayed as a circle with a line (●—). Such a representation helps visualize the object because a dart represents a part of an edge in an Omap. Similarly, in the 2-Gmap (Fig. 3.6c), the 1 and 2-arcs are drawn with short lines, while the 0-arcs are drawn with long lines to help visualize the topological cell (namely the vertices, edges, and faces).

The modeled object (Fig. 3.6a) consists of five quads in 2D. The corresponding 2-Omap (Fig. 3.6b) consists of six cycles of 1-arcs for the five faces plus the outer one. The 1-arcs are oriented anti-clockwise, orienting each face. The 2-arcs link faces along edges. In the 2-Gmap (Fig. 3.6c), a face consists of an alternation of 0-arcs and 1-arcs. Note that the outer face is not represented in a Gmap, and outer nodes are the source of 2-loops.

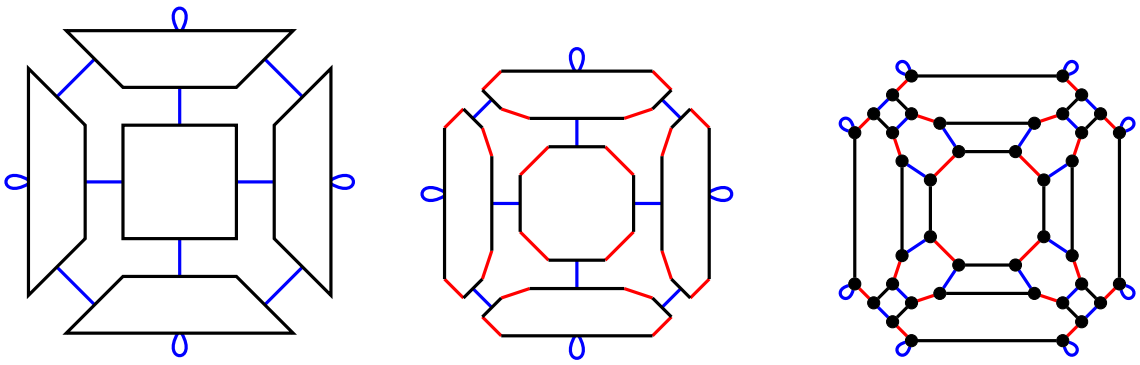
To help construct an intuition about the geometric object represented by such graphs, let us explain how to obtain the representation of such objects as combinatorial maps. The construction detailed here relies on the recursive decomposition of objects into topological cells. The *cellular decomposition* is a top-down approach where we start from the object and decompose it into sub-parts.



(a) (b) (c)

Figure 3.7: Cellular decomposition of an object and construction of the underlying Omap: (a) an oriented 2D object, (b) decomposition of the dimension 2, and (c) the 2-Omap after decomposing the dimension 1.

Example 32 (Object decomposition)**.** The representations of the object of Figure 3.6 as 2-Gmap and 2-Omap result from the cellular decomposition into decreasing dimensions.



(a) (b) (c)

Figure 3.8: Decomposition of an object and construction of the underlying Gmap: (a) decomposition of the dimension 2, (b) decomposition of the dimension 1, and (c) the 2-Gmap after decomposing the dimension 1.

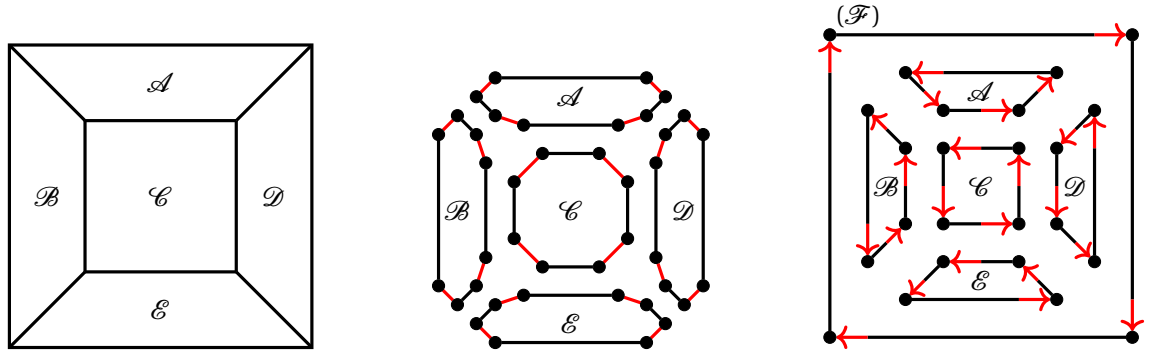
The case of the Omap is illustrated in Figure 3.7. Omaps represent closed-oriented objects. Therefore, in Figure 3.7a, the faces are oriented, including the outer face. All faces share the same orientation. Thus, an edge shared between two faces is oriented in opposite directions in each face. In the decomposition process, we first split the faces along their joint edges, as illustrated in Figure 3.7b. Edges in adjacent faces are pairwise associated be the decomposition of dimension 2. Thus, the arcs added are non-oriented. Then, edges are separated via 1-arcs. The decomposition stops at this step and provides the graph of Figure 3.7c. Here, the arcs follow the orientation of the faces; thus, oriented arcs are added. The nodes of this graph (displayed as ●—) correspond to darts that intuitively represent parts of the edges from the initial object.

In the case of Gmaps, we consider the object open, i.e., there are no outer faces. The decomposition process is similar (Figure 3.8), but orientation is not taken into account, and the 0 dimension is also split. First, the object is split into faces sharing an edge with 2-links (Figure 3.8a). A loop is added when the edge is not shared between two faces, e.g., the edge belongs to the object's boundary. Then, the faces are decomposed into edges via 1-links (Figure 3.8b). Finally, edges are dissociated with 0-links (Figure 3.8c). The resulting nodes are the nodes of the Gmap, and the dimensional relations are the arcs of the Gmap. As we can see from the example, Gmaps require more nodes to represent an object.

The combinatorial models describe topological information about geometric objects. More precisely, they encode topological relations between the object's subpart. In topology-based geometric modeling, these subparts are the object's vertices, edges, faces, and volumes. These subparts are generically called *topological cells*. Within the model of generalized maps, a cell of dimension i, i-cell for short, corresponds to one orbit[1] of the set of all permutations but the permutation of dimension i. Therefore, topological cells can be retrieved as subgraphs induced by a subset of dimensions. These subgraphs are simply called orbits [145, 144, 16, 12]. In an oriented map, the definition of cells as orbits holds for all dimensions except the dimension 0, i.e., for the vertices. Indeed, because oriented maps consider parts of edges as atomic elements, vertices are not directly accessible. The computation of a vertex requires getting an orbit associated with the set of permutations $\{\beta_k \circ \beta_1 \mid 2 \leqslant k \leqslant n\}$ where n is the dimension of the oriented map and β_i is the permutation associated with dimension i. Intuitively, retrieving cells in an oriented map requires

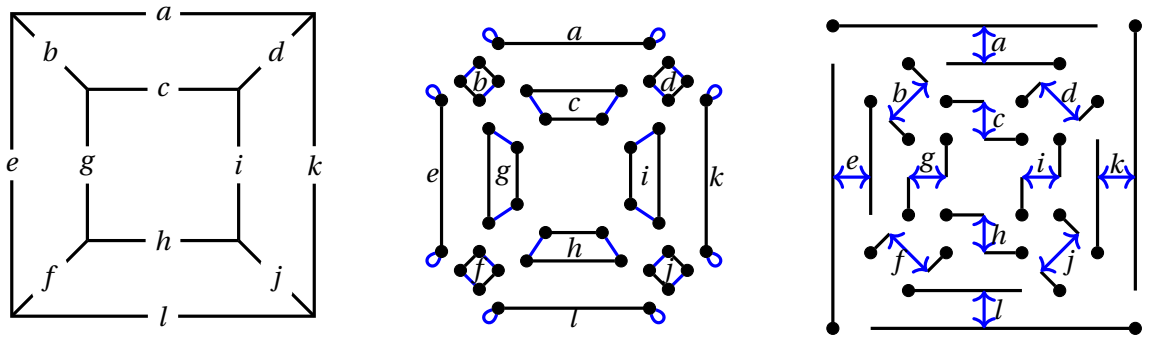
[1] Formally, orbits are the equivalence class induced by the permutations.

computing subgraphs induced by paths. Such subgraphs will be properly defined in Chapter 4 as a generalization of orbits defined in previous works. For now, we reuse the object of Figure 3.6a to illustrate the topological cells.



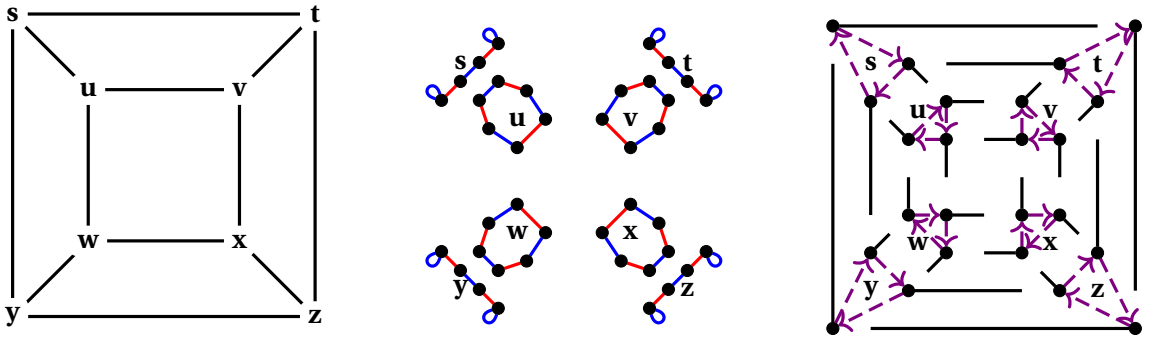
(a) (b) (c)

Figure 3.9: Faces on the object retrieved as 2-cells: (a) the object, (b) 2-cells in the 2-Omap, and (c) 2-cells in the 2-Gmap.



(a) (b) (c)

Figure 3.10: Edges on the object retrieved as 1-cells: (a) the object, (b) 1-cells in the 2-Omap, and (c) 1-cells in the 2-Gmap.



(a) (b) (c)

Figure 3.11: Vertices on the object retrieved as 1-cells: (a) the object, (b) 0-cells in the 2-Omap, and (c) 0-cells in the 2-Gmap.

Example 33 (Topological cells)**.** Let us consider the same object as in Example 31. It is a 2D object, meaning that its topological cells are its vertices, edges, and faces. We will use the standard notation for talking about the permutations associated with Gmaps and Omaps, namelyα_i for

the permutation associated with dimension i in a generalized map and β_i in an oriented map.

Faces (capital calligraphic letters) The faces are illustrated in Figure 3.9a. They correspond to 2-cells in the combinatorial models. In the 2-Omap (Figure 3.9b), the 2-cells correspond to the orbits of the permutation β_1.In the 2-Gmap (Figure 3.9c), the 2-cells correspond to the orbits for the set of permutations $\{\alpha_0, \alpha_1\}$.

Edges (lower case letters in italics) The edges are illustrated in Figure 3.10a. They correspond to 1-cells, i.e., the orbits of the permutation β_2 in the 2-Omap (Figure 3.10b) and the orbits for the set of permutations $\{\alpha_0, \alpha_2\}$ in the 2-Gmap (Figure 3.10c).

Vertices (bold lower case letters) The vertices are illustrated in Figure 3.11a. They correspond to 0-cells. In the 2-Gmap (Figure 3.11c), they are encoded with the orbits for the set of permutations $\{\alpha_1, \alpha_2\}$. In the 2-Omap (Figure 3.11b), the 0-cells correspond to the orbits of the permutation $\beta_2 \circ \beta_1$, depicted with purple dashed arrows.

In this section, we only discussed examples in 2D because they are easier to represent. However, all definitions and constructions are valid in any dimension, and we will see 3D example later.

3.3 Topological consistency on double pushout rules

Given that Gmaps and Omaps are defined as graphs, we formalized modeling operations as graph transformations. We now discuss the design of operations as DPO rules and the preservation of the model consistency. More precisely, we study conditions related to each topological constraint (incident arcs, non-orientation, and cycles) introduced in Section 3.2. Superimposing the appropriate conditions will grant Gmaps (resp. Omaps) consistency preservation. As already argued at the beginning of the chapter, we want to ensure consistency at the rule level to provide feedback to the rule designer, i.e., obtain conditions checkable at design time.

Consistency preservation has been studied in previous work [144], but only sufficient conditions were given. As a result, the syntax analyzer [16, 12] would raise false negatives and consider valid rules as inconsistent. We will give necessary and sufficient conditions to preserve the topological constraints on a local scale in topological graphs. Note that the non-orientation property was not a concern for generalized maps. Imposing that all graphs be non-oriented was sufficient to preserve the non-orientation constraint [144]. With the introduction of combinatorial maps, the study needs to be more general. This study leads us to provide a weaker yet sufficient condition for preserving the non-orientation constraint. For the incident arcs condition, the proof of the consistency preservation remains an extrapolation of the one given in [144].

In this section, we consider i and j to be two dimensions in $\mathbb{D}$.

3.3.1 Incident arcs consistency

We start with the preservation of the incident arcs constraint (Definition 29). Combinatorial graphs play a key role in our framework and will prove essential for the study conducted in Sections 3.3.3 and 3.3.5 on the remaining two constraints.

Recall from Chapter 2 that we consider linear rules, i.e., partial monomorphisms, where the interface is denoted by $\mathrm{L} \cap \mathrm{R}$. The complete associated notation with morphisms is $\mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$. We call preserved the elements of the interface, deleted the elements of L not in the image of $i_\mathrm{L}(\mathrm{L} \cap \mathrm{R})$, and added the elements of R not in the image of $i_\mathrm{R}(\mathrm{L} \cap \mathrm{R})$. The weak incident arcs condition states that preserved nodes should retain their incidence to a dimension, and added nodes should satisfy the incident arcs constraint.

Definition 33 (Weak incident arcs condition)**.** *A rule* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *satisfies the* weak incident arcs condition $\texttt{w-I}_r(i)$ *if it satisfies the two following sub-conditions :*

1. *Any preserved node is the source (resp. target) of an* i*-arc in* L *if and only if it is the source (resp. target) of an* i*-arc in* R.
2. R *satisfies* $\texttt{I}_{(\mathrm{V_R} \setminus \mathrm{V_L}),\mathrm{R}}(i)$*, i.e., any added node is the source of a unique* i*-arc and the target of a unique* i*-arc in* R.

When the weak incident arcs condition is fulfilled, the derivation yields a graph satisfying the incident arcs constraint.

Theorem 1 (Incident arcs preservation)**.** *A rule* r *in the category* $\mathbb{D}$**-Graph** *satisfies the weak incident arcs condition* $\texttt{w-I}_r(i)$ *if and only if for all matches* $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ *on a* $\mathbb{D}$*-graph satisfying* $\texttt{I}_\mathrm{G}(i)$*, the result graph* H *of the direct derivation* $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ *satisfies the incident arcs constraint* $\texttt{I}_\mathrm{H}(i)$.

$$r \models \texttt{w-I}_r(i) \iff (\forall m\colon \mathrm{L} \hookrightarrow \mathrm{G}, \mathrm{G} \models \texttt{I}_\mathrm{G}(i) \implies \mathrm{H} \models \texttt{I}_\mathrm{H}(i))$$

The proof can be summarized as follow. In the direct sense, we consider a node in H and distinguish between cases: the node can be new and added by the rule, or it can also be present in G. If it was present, we distinguish again between the case where its incident i-arc is modified by the rule. For new nodes, the constraint holds from sub-condition 2. For existing nodes, the constraint holds from sub-condition 2 and the properties of a pushout. In the reverse sense, we reason from the origin of the i-arc.

Proof. The first part of this proof ($\Longrightarrow$) has been shown in [144]. Intuitively, regardless of whether preserved nodes are the source of an arc in L (i.e., whether the arc is matched), sub-condition 1 ensures the same status in R, yielding an arc in H. Similarly, sub-condition 2 guarantees the property for added nodes.

($\Longleftarrow$) Suppose the result graph H of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ satisfies the incident arcs constraint $\texttt{I}_\mathrm{H}(i)$, for all matches $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ on a $\mathbb{D}$-graph satisfying $\texttt{I}_\mathrm{G}(i)$. Consider such a graph H and m' the mono $\mathrm{R} \hookrightarrow \mathrm{H}$.

▶ *Sub-condition 1 of the weak incident arcs condition.*

Let v be a preserved node of $\mathrm{L} \cap \mathrm{R}$ that is the source of an i-arc e in L. Then, $m(v)$ is the source of $m(e)$ in G. Suppose this arc is deleted by the application of the rule (otherwise e is in R, which concludes the proof). However, H satisfies $\texttt{I}_\mathrm{H}(i)$ so there exists an i-arc e_H in H of source $m'(v)$. This arc is not in G because $m(v)$ is the source of only one i-arc in G. Thus, an arc e' exists in R such that $m'(e') = e_\mathrm{H}$. By construction the source of e' is v. Therefore, v is the source of an i-arc in R.

The construction holds when taking v as the target of e or inverting L and R. Hence, any preserved node of $L \cap R$ is the source (resp. target) of an i-arc in L if and only if it is the source (resp. target) of an i-arc in R.

▶ *Sub-condition 2 of the weak incident arcs condition.*

Let v be an added node in $V_R \setminus V_L$. Then, $m'(v)$ is an added node in H. Since H satisfies $\mathtt{I}_H(i)$, then an i-arc e_H of source $m'(v)$ exists in H. Since v is not in $L \cap R$, any incident arc of v comes from R, i.e., there exists e in R such that $m'(e) = e_H$ and $s_R(e) = v$. Because e_H is unique, so is e. Thus, v is the source of a unique i-arc in R. The proof holds when replacing source by target, and R satisfies $\mathtt{I}_{(V_R \setminus V_L),R}(i)$.

Thereafter r satisfies the weak incident arcs condition $\mathtt{w\text{-}I}_r(i)$. □

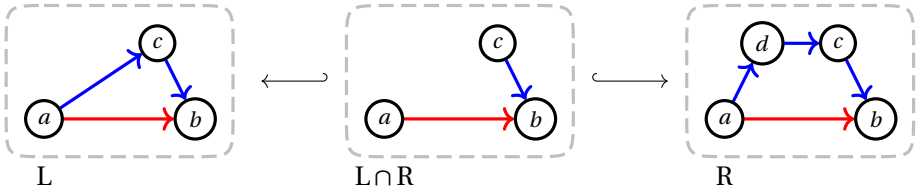


Figure 3.12: A rule r satisfying $\mathtt{w\text{-}I}_r(2)$, the incident arcs condition for the dimension 2.

Example 34 (Incident arcs preservation)**.**

Let r be the rule of Figure 3.12.

- Node a is preserved and the source of a 2-arc in both L and R while not being the target of one.
- Node b is preserved and the target of a 2-arc in both L and R while not being the source of one.
- Node c is preserved, and both the source and the target of a 2-arc in L and R.
- Node d is added, and both the source and the target of a unique 2-arc and R.

Therefore, the rule $r \models \mathtt{w\text{-}I}_r(2)$. Since node d is neither the source nor the target of a 1-arc and R, the rule does not satisfy the condition for the dimension 1.

An example of direct derivation associated with r is given in Figure 3.13. The occurrences of the match and comatch are highlighted in green, and the nodes are mapped as follows: $a \mapsto x$, $b \mapsto w$, $c \mapsto y$, and $d \mapsto z$. The graph G in the bottom left corner satisfies $\mathtt{I}_G(2)$, the incident arcs constraint for the dimension 2. The nodes in G, not in the occurrence of the match, keep their incident arcs nodes in H. The nodes in the occurrence of the comatch satisfy the constraint via the condition on the rule. Thus, $H \models \mathtt{I}_H(2)$.

The gluing condition [115, 57] ensures the applicability of a rule, provided the existence of a pushout complement. As shown in [144], the gluing condition can be statically checked on the rule when applied to combinatorial graphs: a match $m\colon L \hookrightarrow G$ for a rule $r = L \xhookleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$, where G is a $\mathbb{D}$-combinatorial graph, satisfies the dangling condition if and only if L satisfies $\mathtt{I}_{(V_L \setminus V_R),L}(\mathbb{D})$.

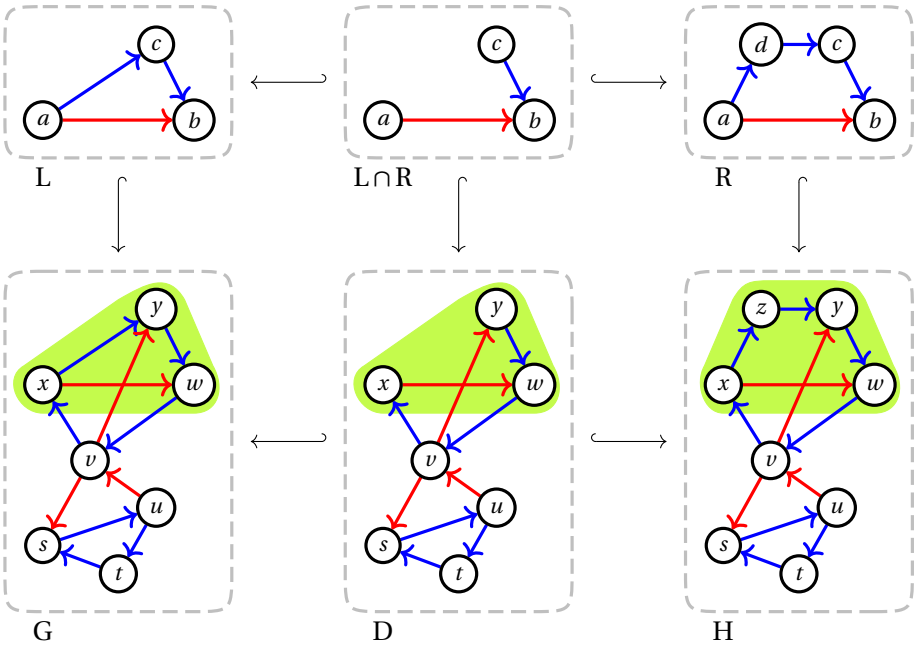


Figure 3.13: Incident arcs preservation for the dimension 2 in a direct derivation.

Fact 1. *For a $\mathbb{D}$-rule applied to a $\mathbb{D}$-combinatorial graph, the dangling condition on any match is equivalent to the incident arcs constraint on the left-hand side of the rule, restricted to the deleted elements (proof in [144]).*

Therefore, we extend the weak incident arcs condition to incorporate the dangling condition aggregating in the incident arcs condition. Once again, this condition is defined on a per-dimension basis.

Definition 34 (Incident arcs condition)**.** *A rule $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ satisfies the* incident arcs condition $\mathtt{I}_r(i)$ *if $r \models \mathtt{w\text{-}I}_r(i)$ and $\mathrm{L} \models \mathtt{I}_{\mathrm{V_L}\setminus\mathrm{V_R},\mathrm{L}}(i)$.*

When working with combinatorial graphs, a match cannot exist if a node in L is the source (or target) of several arcs for one dimension. Such rules can never be applied to a combinatorial graph. Therefore, we impose a further condition: each node should be the source and target of at most one arc per dimension. Note that it suffices to impose the condition on L to obtain it on L∩R. Then, the weak incident arcs condition carries it over to R.

Definition 35 (Combinatorial rule)**.** *A rule $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ in the category $\mathbb{D}$-***Graph*** is a $\mathbb{D}$-*combinatorial rule *if the two following properties hold.*

- *$r \models \mathtt{I}_r(\mathbb{D})$ (Definition 34)*
- *For all nodes v of $\mathrm{V_L}$, there exists at most one arc of source, resp. target, v in $\mathrm{E_L}$.*

Topological formalization of modeling operations relies heavily on combinatorial rules and graphs. Indeed, the incident arcs constraint guarantees the existence and uniqueness of arcs given a label (and a source node). Therefore, we will consider (unless stated otherwise) that the following assumptions hold.

Assumption 2 (Combinatorial graphs)**.** *Graphs under modification belong to the category of combinatorial graphs. This category is the full subcategory of $\mathbb{D}$-***Graph*** whose objects are the $\mathbb{D}$-combinatorial graphs. We write $\mathbb{D}$-***CGraph*** for the category of $\mathbb{D}$-combinatorial graphs.*

Assumption 3 (Combinatorial rule)**.** *Rules are combinatorial rules, transforming a combinatorial graph into a combinatorial graph.*

Before detailing conditions for the preservation of the non-orientation and cycle constraint, we do a little detour, discussing the representation of combinatorial rules. This discussion serves several purposes:

- simplifications in the examples, by allowing the representation of a rule simply from a left-hand side and a right-hand side,
- simplifications in some constructions and proofs, by enforcing some properties on the interface,
- description of rules closer to their practical representation (further discussed in the second part of the dissertation).

3.3.2 Representation of rules

Working with combinatorial rules simplifies the practical description of a rule. Indeed, the node function of the partial mono is sufficient to define the rule. The arc function can be deduced greedily by considering that whenever there is an arc between a pair of mapped nodes before and after the mapping, then the arc is also mapped. Such notation is used to simplify the visual display of rules. However, it is also of practical interest: we can write rules without having to describe any morphism. Node identifiers are enough to specify the preserved, deleted, and added elements.

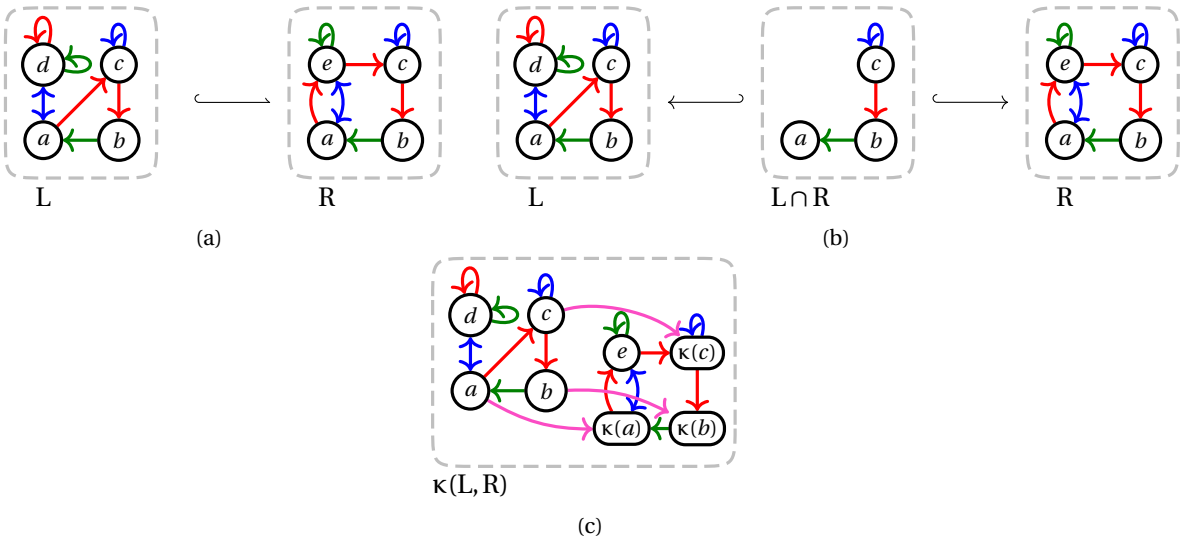


Figure 3.14: Reconstruction of a combinatorial rule from a partial mapping of nodes: (a) a partial mono only described as a node mapping induced by names on the graphs, (b) the reconstructed span, and (c) its joint representation.

Example 35 (Constructing a rule from a partial injection on nodes)**.** Figure 3.14a represents a rule, given as a partial mono, but without actually describing the morphism. Only node identifiers are given. Such identifiers uniquely describe the node component of the rule: nodes a, b, and c are preserved, i.e., in the interface; node d is deleted; node e is added. We could then build the interface as the discrete graph whose nodes are a, b, and c. Such a graph would have no arc,

which complexifies the consistency preservation study. Therefore, we exploit the uniqueness of arcs to add in the interface all arcs which can be found to have an image by both $i_L : L \cap R \hookrightarrow L$ and $i_R : L \cap R \hookrightarrow R$. In the rule of Figure 3.14a, these arcs correspond to the 1-arc from c to b, the 2-loop on c, and the 3-arc from b to c. We obtain the rule of Figure 3.14b.

Therefore, we will consider that the following assumption holds.

Assumption 4 (Preserved arcs)**.** *In a combinatorial rule (Definition 35), if there is an i-arc of source v_s and target v_t in both* L *and* R*, then it is a preserved arc of* $L \cap R$.

Under such an assumption, a rule can be represented as a single graph, like in the integrated notation of rules [99]. Here, we use a slightly different construction. We consider the disjoint union of both the left-hand and right-hand sides, plus a mapping on nodes signified by arcs labeled with a fresh symbol κ. Those arcs are considered oriented from the left-hand side node to its right-hand side counterpart. This graph is called the joint representation of the rule.

Definition 36 (Joint representation of a rule)**.** *Let κ be a fresh symbol, i.e., $\kappa \notin \mathbb{D}$. The* joint representation *of a rule $r = L \xleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ in $\mathbb{D}$-***Graph***, is the graph $\kappa(L,R) = (V_\kappa, E_\kappa, s_\kappa, t_\kappa, l_\kappa)$ in $(\mathbb{D} \cup \kappa)$-***Graph*** *such that:*

- $V_\kappa = V_L \sqcup V_R$,
- $E_\kappa = E_L \sqcup E_R \sqcup V_{L\cap R}$,
- $$s_\kappa : e \mapsto \begin{cases} s_L(e) & \text{if } e \in E_L \\ s_R(e) & \text{if } e \in E_R \\ i_L(v) & \text{if } e \in V_{L\cap R} \end{cases} \quad t_\kappa : e \mapsto \begin{cases} t_L(e) & \text{if } e \in E_L \\ t_R(e) & \text{if } e \in E_R \\ i_R(v) & \text{if } e \in V_{L\cap R} \end{cases} \quad l_\kappa : e \mapsto \begin{cases} l_L(e) & \text{if } e \in E_L \\ l_R(e) & \text{if } e \in E_R \\ \kappa & \text{if } e \in V_{L\cap R} \end{cases}$$

When working with the joint representation $\kappa(L,R)$ of a rule $r = L \xleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$, we also store the partition of V_κ into $V_L \sqcup V_R$ such that we know whether a node belongs to the left-hand side or the right-hand side. For i-arcs, we can deduce the side from the side of their source and target. Indeed, only κ-arcs are incident to nodes in distinct parts of the rule.

Example 36 (Joint representation of a rule)**.** Figure 3.14c illustrates the joint representation of the rule studied in Example 35. In this representation, any node from $L \cap R$ appears twice, once from L and once from R. We use the identifier $\kappa(x)$ for the representant of a node of K from R. For instance, node a, resp. $\kappa(a)$, in Figure 3.14c corresponds to node a in L, resp. R, from Figure 3.14b. Any such node yield the addition of a κ-arc from its L representant to its R representant. Since $L \cap R$ contains the three nodes a, b, and c (see Figure 3.14b), the graph $\kappa(L,R)$ contains nodes a, $\kappa(a)$, b, $\kappa(b)$, c, and $\kappa(c)$. These nodes are linked with κ-arcs from a to $\kappa(a)$, from b to $\kappa(b)$, and from c to $\kappa(c)$. All nodes not in K keep their identifiers, such as nodes d and e. Note that all i-arcs, i.e., non $kappa$-arcs, correspond to the arcs of either L or R.

We retrieve the original rule from the joint representation of a rule by removing the κ-arcs and splitting the set of nodes accordingly. The other arcs are added to the rule's left-hand or right-hand side based on the side of their source and target.

We will use the joint representation of a rule for two purposes. First, the joint representation allows computing rule isomorphism as graph isomorphism. Secondly, the joint representation

can serve as an intermediate structure for automatic rules generation. Both purposes are topics for the second part of this dissertation, and we will not use the joint representation in the first part of this dissertation. Nonetheless, the precise definition of this representation requires the more formal algebraic definition of graphs, which we will try to use as rarely as possible when dealing with geometric modeling. Therefore, we believe here and now to be the right place and time to define this joint representation properly.

We now resume the construction of consistent rules. With the benefit of the incident arcs constraint and condition, we will present necessary and sufficient conditions to preserve both the non-orientation and cycle constraints. A watchful reader will notice that we could impose a less restrictive constraint. Indeed, the incident arcs constraint and condition for a dimension i are enough to deal with the non-orientation property on this same dimension. Similarly, the incident arcs constraint for the dimensions i and j are sufficient for the cycle property on these dimensions.

3.3.3 Non-orientation consistency

To preserve the non-orientation (Definition 30), we require that any added or deleted ars is modified with its reverse arc.

Definition 37 (Non-orientation condition)**.** *A $\mathbb{D}$-combinatorial rule $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ satisfies the* non-orientation condition $\mathtt{O}_r(i)$ *if any deleted, resp. added, i-arc of has a unique reverse i-arc that is also deleted, resp. added.*

In other words, the non-orientation condition boils down to only having oriented arcs in the interface $\mathrm{L}\cap\mathrm{R}$. We now give the corresponding theorem.

Theorem 2 (Non-orientation preservation)**.** *A $\mathbb{D}$-combinatorial rule r satisfies the non-orientation condition $\mathtt{O}_r(i)$ if and only if for all matches $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ with $\mathrm{G} \in \mathbb{D}$-**CGraph** and $\mathrm{G} \models \mathtt{O}_\mathrm{G}(i)$, the result graph H of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ satisfies the non-orientation constraint $\mathtt{O}_\mathrm{H}(i)$.*

$$r \models \mathtt{O}_r(i) \iff (\forall m\colon \mathrm{L} \hookrightarrow \mathrm{G}, \mathrm{G} \models \mathtt{O}_\mathrm{G}(i) \implies \mathrm{H} \models \mathtt{O}_\mathrm{H}(i))$$

Proof. The proof is similar to the preservation of the incident arcs constraint. In the direct sense, we consider the status of an arc, i.e., new and added by the rule, in the occurrence of the match but not modified, or outside the occurrence of the match and already present in the initial graph. From the status of the arc, we determine the plausible status for a reverse arc and use the non-orientation condition to conclude. In the reverse sense, we consider new arc in H and removed arcs in G and show that their reverse counterpart is also new or removed, exploiting the constraint on the graph and the monic property of the match or comatch. The complete proof can be found in Appendix A.1, extracted from [139]. □

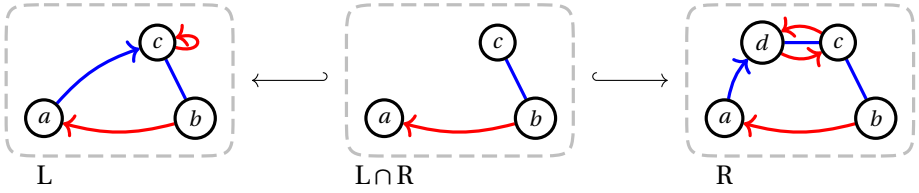


Figure 3.15: A rule r satisfying $\mathtt{O}_r(1)$, the non-orientation condition for the dimension 1.

Example 37 (Non-orientation preservation)**.**

Let r be the rule of Figure 3.15. For readability purposes, oriented arcs are bent while non-oriented arcs which are not of interest are drawn with straight, arrowless lines.

- The deleted 1-loop on node c is its own inverse.
- The added 1-arc from node c to node d are reciprocally their unique inverse.
- The invert-free 1-arc from node b to node a is preserved.

Therefore, the rule $r \models \mathtt{O}_r(1)$. Since neither the deleted 2-arc from node a to node c, nor the added 2-arc from node a to node c admits a reverse arc, r does not satisfy $\mathtt{O}_r(2)$.

An example of direct derivation associated with r is given in Figure 3.16. The occurrences of the match and comatch are highlighted in green. The nodes are mapped as follows: $a \mapsto x$, $b \mapsto w$, $c \mapsto y$, and $d \mapsto z$. The graph G in the bottom left corner satisfies $\mathtt{O}_G(\{1,2\})$, the non-orientation constraint for the dimensions 1 and 2. Since $r \models \mathtt{O}_r(1)$, it holds that $H \models \mathtt{O}_H(1)$. Besides, H possesses two arcs without reverse arc: the 2-arc from node x to node z and the 2-arc from node y to node x. Therefore, $H \not\models \mathtt{O}_H(2)$, which is consistent with the fact that $r \not\models \mathtt{O}_r(2)$.

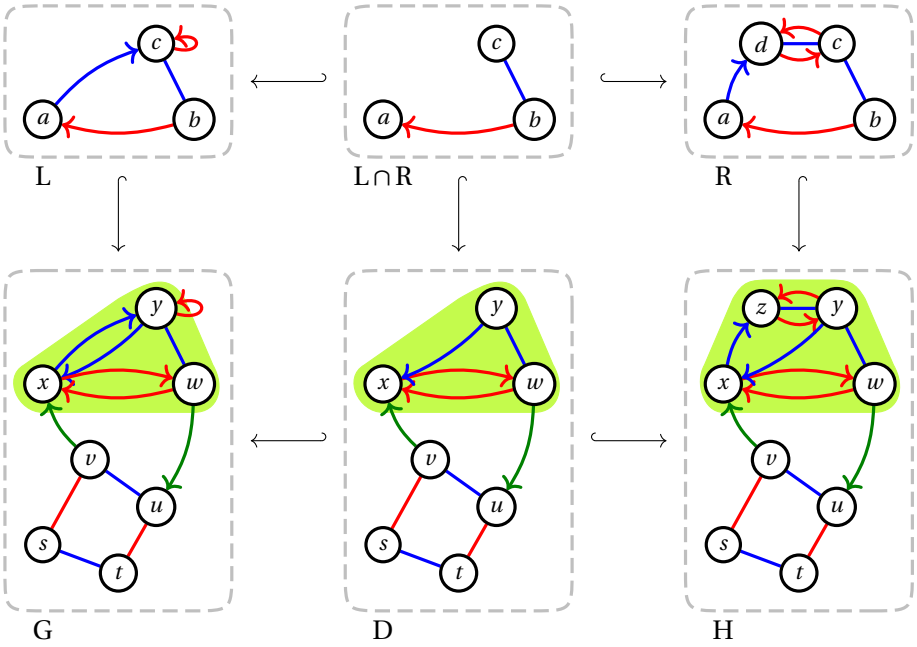


Figure 3.16: Non-orientation preservation for the dimension 1 in a direct derivation.

In practice, dimensions are considered either oriented or not, depending on whether the corresponding permutation is an involution. Thus, a reachable solution is to write rules where the arcs are non-oriented whenever the dimension is non-oriented in the model.

3.3.4 Dealing with the cycle constraint

Finally, the cycle constraint (Definition 31) needs a comparable condition on transformation rules to acknowledge the rules that guarantee consistency. The main issue when dealing with the cycle constraint is the description of neighboring elements. The preservation of the first two topological constraints considered arcs always incident to nodes in the occurrence of the match. Since rules

may only partially match cycles, we discuss constructions and properties that will help us define a necessary and sufficient condition to preserve the cycle constraint.

Recall from Lemma 7 that the order of the dimensions for the cycle constraints does not matter in a combinatorial graph when considering the constraint on the whole graph.

Rule completion

First, since the cycle constraint characterizes cycles of length 4, if there is an iji-path or a jij-path in L, we can add the missing arc to make a cycle. In other words, if there is an iji-path p in the left-hand side L of a rule r and we apply r to a graph G that satisfies $\mathfrak{C}_G(i,j)$ via the match morphism $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ then there is a j-arc of source $m(t(p))$ and target $m(s(p))$ in G. If not already in the rule interface, this j-arc can be added to L∩R without modifying the result of the direct transformation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$. For instance, we can add the missing j-arcs to the rule of Figure 3.17a and get the rule of Figure 3.17b. The application of either rule results in the same graph. However, it is easier to use the latter construction because it clarifies the rules' cycles. Consequentially, when considering the cycle constraint in a rule, we first impose the completion of every iji-path and jij-path (in L and R). We call this operation the ij-completion of the rule and assume that every rule is ij-completed.

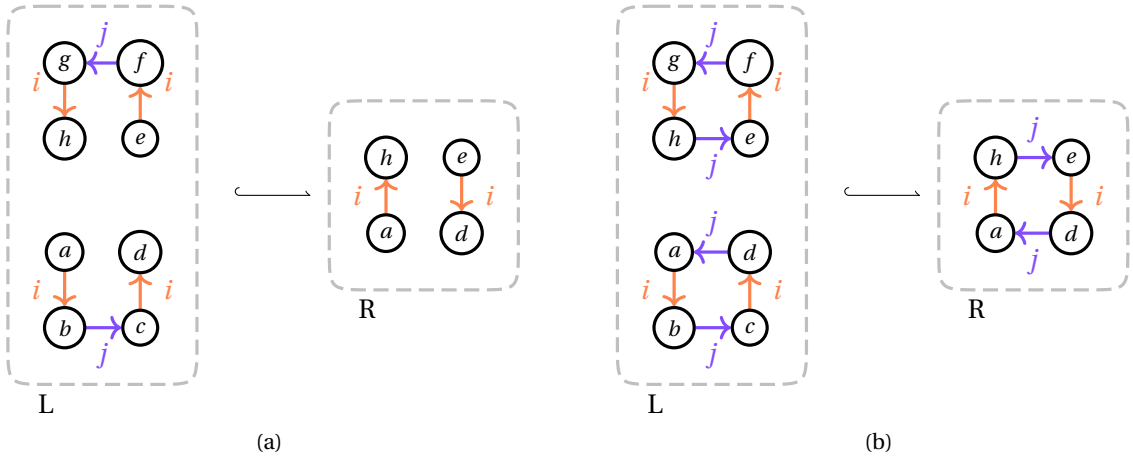


Figure 3.17: ij-completion of a rule: the missing j-arcs in rule (a) are added to the rule (b), revealing the $ijij$-cycle in R.

Assumption 5. *Every rule is ij-completed.*

Alternating paths

Given a rule r and a match $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$, an $ijij$-cycle in G is split into (possibly empty) elements of $\mathrm{G} \setminus m(\mathrm{L})$, $m(\mathrm{L} \cap \mathrm{R})$, and $m(\mathrm{L} \setminus \mathrm{R})$. The elements of $\mathrm{G} \setminus m(\mathrm{L})$ are not matched by the rule and are left unchanged. The elements of $m(\mathrm{L} \cap \mathrm{R})$ belong to the rule interface and are not modified. Therefore, the only part that raises concern consists of the elements in $m(\mathrm{L} \setminus \mathrm{R})$. If we guarantee that for each path in $m(\mathrm{L} \setminus \mathrm{R})$ coming from the cycle, a similar path exists in $m'(\mathrm{R} \setminus \mathrm{L})$, by concatenation with elements of $m'(\mathrm{R} \cap \mathrm{L})$ and elements of $\mathrm{H} \setminus m'(\mathrm{R})$, we reconstruct a cycle in H.

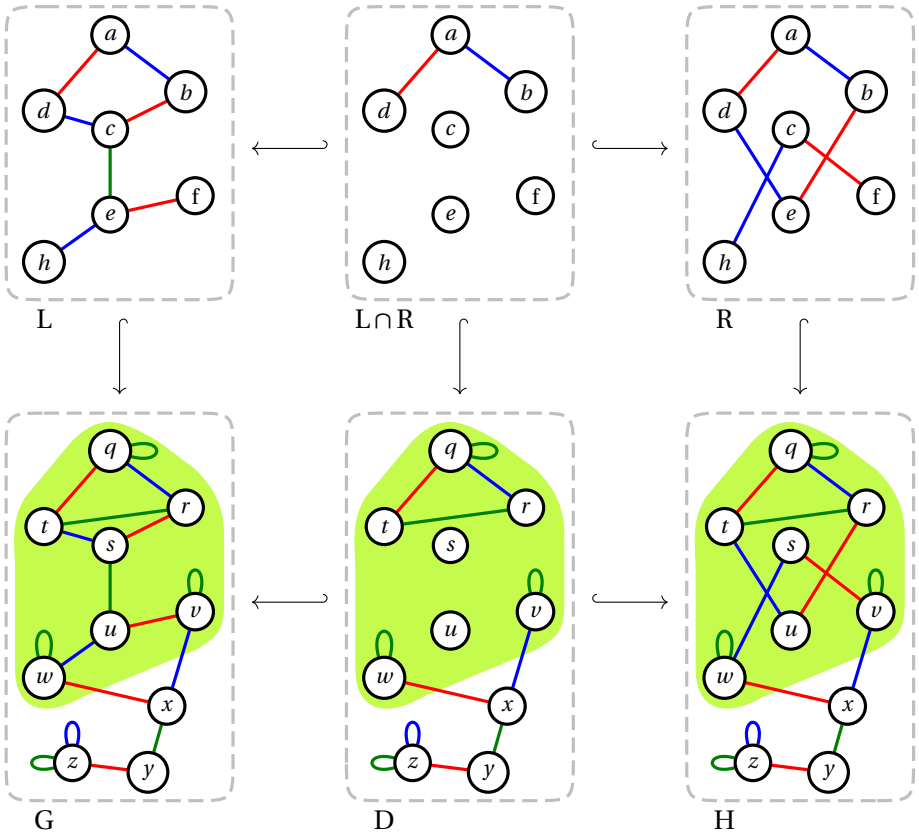


Figure 3.18: A rule preserving the cycle constraint for the dimensions 1 and 2.

Example 38 (Intuition for cycle preservation)**.**

Figure 3.18 illustrates a rule that flips two nodes linked with a 3-arc and removes the arc. In this example, we only used non-oriented arcs to manipulate graphs with a reasonable number of arcs and nodes. The complete match can be deduced from its node component $\{a \mapsto q, b \mapsto r, c \mapsto s, d \mapsto t, e \mapsto u, f \mapsto v, h \mapsto w\}$, highlighted in green. The rule's application essentially exchanges the roles of nodes s and u between the top and bottom 1212-cycle from graph G to graph H, while deleting the 3-arc. In the right-hand side of the rule, nodes a, b, d and e belongs to a cycle. The question is whether the images of c, f, and h also belong to a cycle in H. A 12-path from f to h exists in both L and R. Thus, if a path's image belongs to a cycle in G, the unmatched part can be concatenated with the image of the path from R. The derivation displayed in Figure 3.18 modifies a graph where the path belongs to such a cycle. As a result, we obtain a cycle in H. This example provides intuition to preserve cycles with rules, explained more thoroughly in this section.

Formally, elements in L that can be matched to an $ijij$-cycle in G are paths labeled by a word on the alphabet $\{i, j\}$ such that two consecutive letters are distinct. We call (i, j)-*alternating* such paths. We want to ensure that if L contains an (i, j)-alternating path with all arcs in $\mathrm{E_L} \setminus \mathrm{E_R}$, R also has an (i, j)-alternating path with the same source, target, and label. What we want to consider are only the longest paths. Since L may contain cycles with all arcs in $\mathrm{E_L} \setminus \mathrm{E_R}$, we need to enforce non-overlapping conditions to consider maximal elements. We say a path *overlaps* if it contains an arc twice (note that it may include a node several times). We call optimal paths such paths.

Definition 38 (Optimal path)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be a* $\mathbb{D}$*-combinatorial rule. We say the path*

$p = v_s \rightsquigarrow v_t$ *is* (i,j)-optimal *in* L *(resp. in* R*) if it satisfies the following properties:*

1. *The path p is an (i,j)-alternating path (i.e., path labeled by a word on the alphabet $\{i,j\}$ such that two consecutive letters are distinct).*
2. *All arcs of p are deleted (resp. added).*
3. *The path p does not overlap.*
4. *The path p is maximal (with respect to the inclusion of paths) between paths that satisfy the first* 3 *properties.*
5. *The path p does not belong to an $ijij$-cycle in* L *(resp. in* R*).*

Example 39 (Optimal path)**.** Consider the left-hand side of the rule from Figure 3.18. The 1212-cycle $adcb$ forms a $(1,2)$-alternating path. This path is non-overlapping and maximal. However, it is not optimal as it contains interface arcs (the arc between a and d and the one between b and a). Restricted to the 21-path dcb, it is still not optimal as it is part of an 1212-cycle. The 132-path $bceh$ is labeled with three distinct dimensions. Therefore, it is not an alternating path, and a fortiori is not an optimal path. Conversely, the 131-path $fecb$ is a maximal non-overlapping $(1,3)$-alternating path. As it contains only deleted arcs and is not part of a 1313-cycle, the path is $(1,3)$-optimal.

Note that any optimal path satisfies the following straightforward property: if $p = v_s \rightsquigarrow v_t$ is an (i,j)-optimal path in a $\mathbb{D}$-combinatorial rule $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$, then v_s and v_t are distinct interface nodes of $\mathrm{L} \cap \mathrm{R}$.

Consider again the 1212-cycle of source v in the graph G from Figure 3.18. The cycle can be divided into an optimal path from v to w (the 12-path going through u) and a path from w to v in $\mathrm{G} \setminus m(\mathrm{L})$ (the 12-path going through x). For any optimal path in L, we impose a path with the same source, target, and label in R and call coherent such paths.

Definition 39 (Coherence)**.** *Let $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ be a $\mathbb{D}$-combinatorial rule. An optimal path in* L *(resp. in* R*) is* coherent *with an optimal path in* R *(resp. in* L*) if they share the same source, target, and label. An optimal path in r is* strongly coherent *if it admits a unique, coherent optimal path in the other side of the rule.*

To guarantee that a combinatorial rule preserves the cycle constraint, we impose that every optimal path is strongly coherent.

Definition 40 (Cycle condition)**.** *A $\mathbb{D}$-combinatorial rule $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ satisfies the* cycle condition $\mathtt{C}_r(i,j)$ *if it satisfies the following sub-conditions:*

1. *Any (i,j)-optimal path in r is strongly coherent (Definition 39).*
2. *Any preserved node that is the source of an i-arc (resp. j-arc) in $\mathrm{E_L} \setminus \mathrm{E_R}$ is the source of an i-arc (resp. j-arc) in* R *that either belongs to an $ijij$-cycle or to an (i,j)-optimal path.*
3. *Any added node is the source of an i-arc (resp. j-arc) in* R *that either belongs to an $ijij$-cycle or to an (i,j)-optimal path.*

A $\mathbb{D}$-combinatorial rule r satisfying the cycle condition $\mathtt{C}_r(i,j)$ contains no (i,j)-optimal path that consists of a single arc. Indeed, such an arc would have to be a strongly coherent optimal path and, therefore, would be in $\mathrm{L} \cap \mathrm{R}$, which contradicts the optimality of the path.

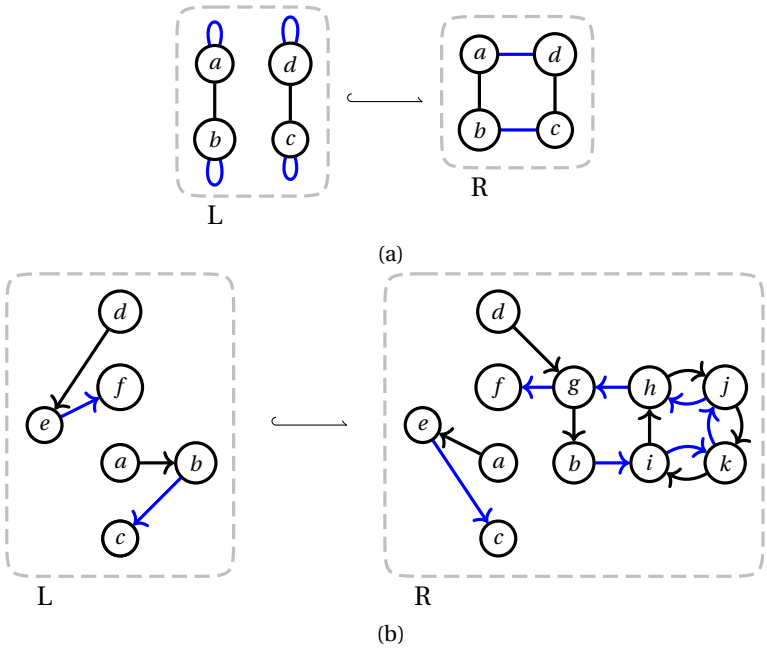


Figure 3.19: Two rules that satisfies the cycle condition $\mathtt{C}_r(0,2)$.

Example 40 (Cycle condition)**.**

One can easily verify that the rule from Figure 3.18 satisfies the cycle condition of Definition 40 for the dimensions 1 and 2. We provide more examples in Figure 3.19, where arcs are still considered non-oriented.

The rule of Figure 3.19a transforms two 0202-cycles, where the 2-arcs are loops, into a single 0202-cycle on four nodes. In L, each node is the source of an 0202-cycle, L admits no $(0,2)$-optimal path. Note that R does not add any $(2,2)$-optimal path. All nodes have their incident 2-arcs in L deleted but are the source of a 2-arc that belongs to a cycle in R. The reader familiar with the manipulation of generalized maps will have recognized the sewing operation for non-degenerate edges on 2-Gmaps. Indeed, each connected component of L encodes a non-degenerate 2-free edge transformed into a single 2-sewn edge in R.

In the rule of Figure 3.19b, L consists of two $(0,2)$-optimal paths. The path abc is coherent in L as a $(0,2)$-optimal path between a and c also exists in R. Since the paths are unique, both are strongly coherent. Similarly, the path def in L is strongly coherent. All nodes of L are interface nodes with deleted 0-arcs and 2-arcs. Nodes a, d, and e are the source of arcs that belong to an optimal path in R. Node b is the source of a 2-arc that belongs to a 0202-cycle in R. The added nodes nodes h, i, j, and k are the sources of a 0-arc and a 2-arc that belong to a 0202-cycle in R. Finally, an added node can also be the source of an arc that belongs to optimal paths. For instance, the 2-arc of source g belongs to an optimal path.

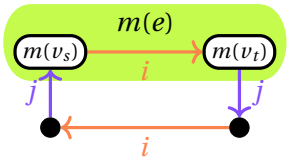
3.3.5 Cycle consistency

Before detailing the theorem for cycle preservation, we will provide a preliminary result on the size of optimal paths in a coherent rule.

Lemma 8. *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an* ij*-completed rule such that for all matches* $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ *with* $\mathrm{G} \in \mathbb{D}$-**CGraph** *and* $\mathrm{G} \models \mathtt{C}_\mathrm{G}(i,j)$*, the resulting graph* H *of the direct derivation* $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ *satisfies the cycle constraint* $\mathtt{C}_\mathrm{H}(i,j)$*. Then, any* (i,j)*-optimal path in* L *is of size* 2*.*

Proof. Let $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ be an ij-completed rule, and G and H be combinatorial graphs in $\mathbb{D}$-**CGraph** such that $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ for a match $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ and a comatch $m'\colon \mathrm{R} \hookrightarrow \mathrm{H}$. Let us assume that $\mathrm{G} \models \mathtt{C}_\mathrm{G}(i,j)$. We further assume that all matches $q\colon \mathrm{L} \hookrightarrow \mathrm{G}'$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{C}_{\mathrm{G}'}(i,j)$, we have $\mathrm{H}' \models \mathtt{C}_{\mathrm{H}'}(i,j)$ where $\mathrm{G}' \Rightarrow^{r,q} \mathrm{H}'$. In particular, $\mathrm{H} \models \mathtt{C}_\mathrm{H}(i,j)$.

Suppose an (i,j)-optimal path exists in L with deleted elements in $\mathrm{L}\setminus\mathrm{R}$ of size 1. Without loss of generality, we can suppose there is an i-arc e of source v_s and target v_t in L such that v_s is the target of no j-arc and v_t is the source of no j-arc. When matched to G, the $ijij$-cycle contains $m(e)$ and an (i,j)-alternating path $p = m(v_t) \overset{jij}{\rightsquigarrow} m(v_s)$ with all arcs in $\mathrm{G}\setminus m(\mathrm{L})$. Below, the arc in $m(\mathrm{L}\setminus\mathrm{R})$ is highlighted in green.



By construction, p is also in H. Since H satisfies the cycle constraint $\mathtt{C}_\mathrm{H}(i,j)$, p can be extended to an $ijij$-cycle with an i-arc of source $m'(v_s)$ and target $m'(v_t)$. As r is a $\mathbb{D}$-combinatorial rule and H is a $\mathbb{D}$-combinatorial graph, this i-arc is in $m'(\mathrm{R})$. From the assumption on preserved arcs (Assumption 4), this contradicts that e is in $\mathrm{L}\setminus\mathrm{R}$.

Furthermore, an (i,j)-optimal path can not be of size greater than 4 as there exists no (i,j)-alternating path of size greater than 4 without duplicate arcs in a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{C}_\mathrm{G}(i,j)$. The ij-completion of the rule prevents it from being of size 3. Finally, the definition of optimality prevents it from being of size 4.

Therefore, any optimal path is of size 2. □

Exploiting this characterization, we show that the cycle condition (Definition 40) on a combinatorial rule is necessary and sufficient to guarantee that the rule's application on a combinatorial graph satisfying the cycle constraint (Definition 31) results in a graph that also satisfies the cycle constraint.

Theorem 3 (Cycle preservation)**.** *A* $\mathbb{D}$*-combinatorial rule* r *satisfies the cycle condition* $\mathtt{C}_r(i,j)$ *if and only if for all matches* $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ *with* $\mathrm{G} \in \mathbb{D}$-**CGraph** *and* $\mathrm{G} \models \mathtt{C}_\mathrm{G}(i,j)$*, the graph* H*, result of the direct derivation* $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ *satisfies the cycle constraint* $\mathtt{C}_\mathrm{H}(i,j)$*.*

$$r \models \mathtt{C}_r(i,j) \iff (\forall m\colon \mathrm{L} \hookrightarrow \mathrm{G}, \mathrm{G} \models \mathtt{C}_\mathrm{G}(i,j) \implies \mathrm{H} \models \mathtt{C}_\mathrm{H}(i,j))$$

Proof. The main reasoning of the proof is similar to the previous two theorems about the incident arcs and the non-orientation property: we analyze the status of the various elements. The study

is more challenging because only a part of the cycle may be modified, which is taken care of by the strong coherence of the optimal paths. The complete proof, presented in [139], is given in Appendix A.2. □

Theorems 1, 2, and 3 express consistency preservation as simple syntactic conditions on the rule. These conditions can be verified by simple graph traversals, as we will see in Chapter 4. Therefore, we can automatically verify rules at design time, i.e., while a user writes them. These verifications are similar to the feedback provided by modern Integrated Development Environments when programming. In our case, we are writing rules rather than code. Our approach is somewhat different from standard approaches for consistency preservation in graph rewriting, which we review in the next section.

3.4 Consistency preservation in graph rewriting

This section presents standard solutions used in graph rewriting to ensure consistency preservation. In particular, we discuss negative and positive application conditions and their extension with nested application conditions.

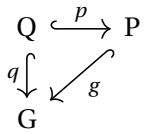
3.4.1 Graph constraints and application conditions

Constraints encode properties on the graph that we expect to preserve through transformations, while conditions denote properties on the rules that allow for the preservation of the constraints.

Graph constraints

When graphs are used for modeling purposes, any graph is usually not admissible. A standard solution to describe admissible graphs relies on constraints that should be satisfied for the graph to be considered consistent with the model. If requested and forbidden structures can specify constraints, the consistency can be studied categorically with application conditions as first introduced in [54]. Exploiting that the algebraic approaches to graph rewriting modify an object locally, application conditions request the existence or non-existence of a given subgraph in the host graph. If this (non-)existence is fulfilled, the production rule can adequately be applied.

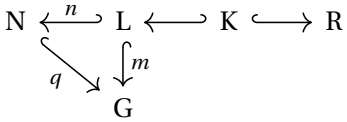
Constraints and conditions are defined via morphisms on graphs. The construction of these two notions is identical, to the point that some authors only speak about conditions. We prefer to keep the two terms such that constraints refer to a graph while conditions refer to a rule. A positive constraint states that a structural part P should exist in a graph G, subject to the existence of a smaller subgraph Q. These constraints are called conditional constraints in [99], or positive atomic constraints [57, Chap. 3] where the term positive constraint encapsulates boolean formulas over atomic constraint. A positive constraint consists of a monomorphism $p\colon \mathrm{Q} \hookrightarrow \mathrm{P}$. It is usually written $c = \forall(\mathrm{Q}, \exists \mathrm{P})$. A graph G satisfies c, written $\mathrm{G} \models c$, if, for all monomorphism $q\colon \mathrm{Q} \hookrightarrow \mathrm{G}$, a monomorphism $g\colon \mathrm{P} \hookrightarrow \mathrm{G}$ exists such that the following diagram commutes.



Negative constraints can be defined from positive constraints by stating that the monomorphism $g\colon \mathrm{P} \hookrightarrow \mathrm{G}$ should not exist. We can twist this definition by replacing Q with the empty graph, which is initial in **Graph** (see Definition 6 and Proposition 2 in Chapter 2). Since an initial graph admits a unique morphism to every object in the category, we are assessing whether the structural part P exists in G. In the negative form, the constraint specifies a forbidden pattern written $\neg$P. We introduced the notations $\forall(\mathrm{Q}, \exists \mathrm{P})$ and $\neg$P. which will be used again in Chapter 5.

Application condition

The diagram defines a positive application condition when Q is the left-hand side of a rule and G is the graph to be modified. If the application of a rule requires a graph pattern to be absent, it is called a negative application condition (or NAC) [87]. We use NACs to clarify how such constraints on graphs yield conditions on rules. A NAC corresponds to a graph N which extends L, i.e., such that there exists a morphism $n\colon \mathrm{L} \hookrightarrow \mathrm{N}$. The elements in N, but not in $n(\mathrm{L})$, form the unwanted or forbidden structure. A rule with a NAC is applicable to a graph if the occurrence of its left-hand side cannot be extended to the full NAC. Formally, a rule $r = \mathrm{L} \hookleftarrow \mathrm{K} \hookrightarrow \mathrm{R}$ with a NAC $n\colon \mathrm{L} \hookrightarrow \mathrm{N}$ is applicable to G via the match $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ if and only if there exists no morphism $q\colon \mathrm{N} \hookrightarrow \mathrm{G}$ such that the following diagram commutes.

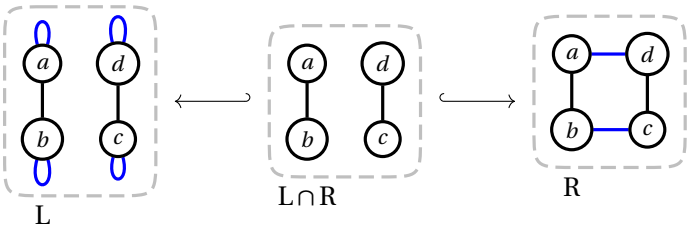


In [87], the theory is developed with arbitrary morphism but, as shown in [89], considering only monomorphisms does not reduce the expressiveness. In any case, the NAC should be matched injectively, i.e., $q\colon \mathrm{N} \hookrightarrow \mathrm{G}$ should be a monomorphism to avoid inconsistency of the NAC satisfiability. For instance, injectivity prevents the forbidden structure from overlapping with the occurrence of the match (see [87]). NACs restrict the applicability of a rule if a structure is already present. For instance, NACs can be used to prevent duplicating some structures: consider a rule that creates an arc between two nodes. Using a NAC, we can proscribe this rule's application if an arc already exists between the two arcs. Application conditions and constraints share the same formal definition but serve different purposes. Constraints relate to the objects, restricting the set of all objects to those which satisfy some properties. Conditions relate to rules (more precisely to matches and comatches), restricting the rule's applicability to the cases that satisfy some properties.

Application conditions can be used to preserve conditions on graphs, i.e., to enforce invariants. Invariants are typically studied for a given set of constraints and a given graph transformation system (i.e., a given set of rules) [99, Chap. 4]. The goal is to build the application condition of each rule to prevent any rule application from producing an inconsistent graph. For a given pair (rule, constraint), the construction of an application condition is a two-step process [56, 99]. First, the constraint is transformed into a right application condition, i.e., an application condition on the rule's right-hand side. Secondly, the right application condition is transformed (or shifted) into an equivalent left application condition. We now describe a typical workflow for obtaining a left application condition.

3.4.2 Obtaining a constraint preserving rule

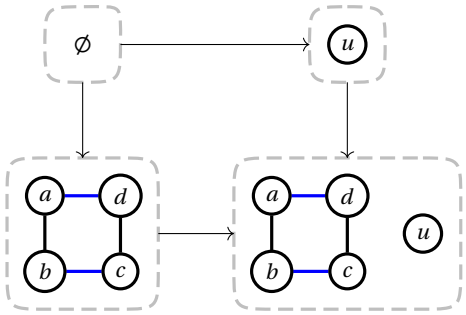
Let us discuss how one can obtain constraint-preserving rules with application conditions. All formal constructions for the shifts of application conditions can be found in [91] or [58]. A more intuitive explanation may be found in [99, Chap. 4]. Consider the following rule, which sews two 2-free edges, as discussed in Example 40.



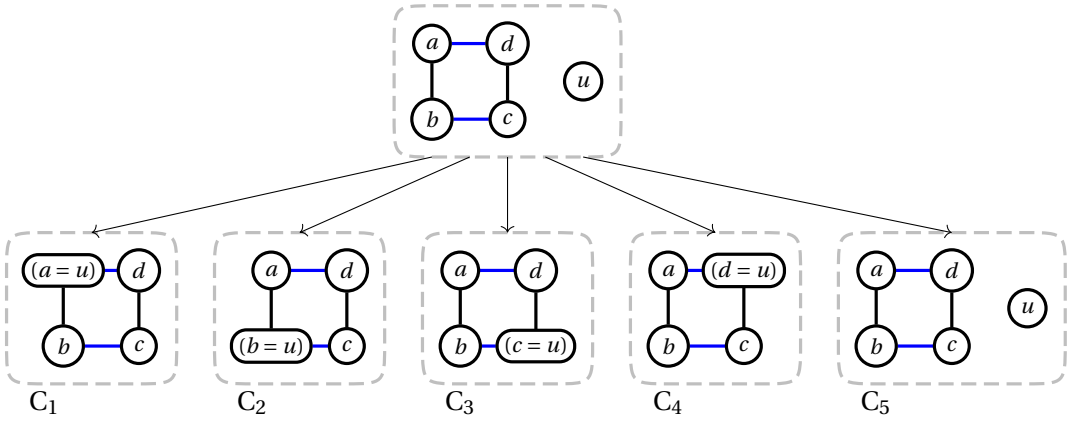
If we consider the incident arc constraint (see Definition 29) for a dimension i globally on the graph, we can describe the constraint as a multiplicity 1 on the typed graph. Multiplicities can be treated as graph constraints and preserved with application conditions. A multiplicity constraint corresponds to several sub-constraints, similar to our constraints $\mathtt{I}_1$ to $\mathtt{I}_4$. The constraint $\mathtt{I}_1$ states the existence of an i arc for a node. Depending on whether the arc is a loop, we obtain two sub-constraints. One constraint for the dimension 2 is $\forall(u),\ \exists(u \text{—} v)$.

The first step is to determine the situations where the application of the rule could lead to a constraint violation. Intuitively, we need to find all overlaps of the constraint with the rule's right-hand side. Formally, the pushout of the constraint and the rule's right-hand side along the initial object in the category is computed. Then, all epimorphisms from the pushout object such that the composition with the pushout morphisms yield monomorphism are considered. Intuitively, this means merging elements of the constraint with elements of the rule's right-hand side without merging elements within the constraint or the rule's right-hand side.

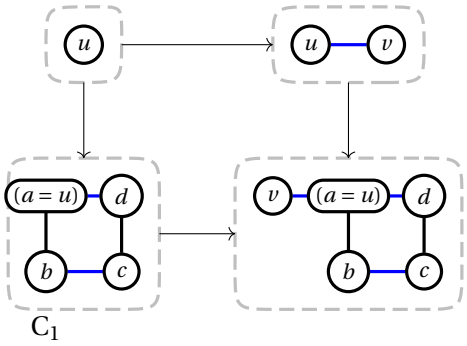
We obtain the following pushout.



From this pushout, we derive five epimorphisms.

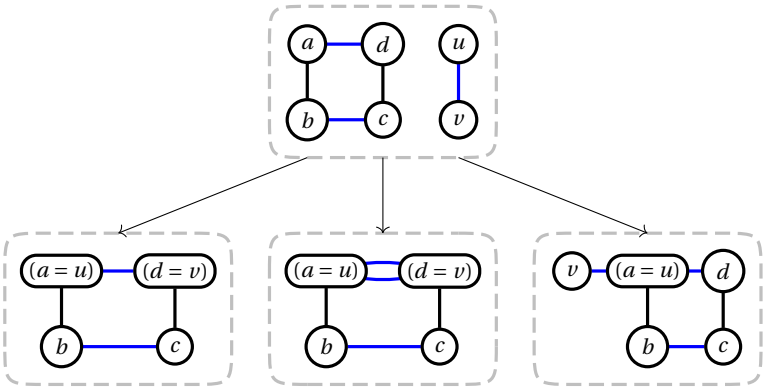


Situations where the right morphism and the morphism from the right-hand side to the constructed object do not admit a pushout complement are discarded. In our cases, all configurations are kept. The process is now iterated with the second part of the application condition, using the first part to build the pushout. Considering the graph C_1 from the first step, we obtain the following pushout.



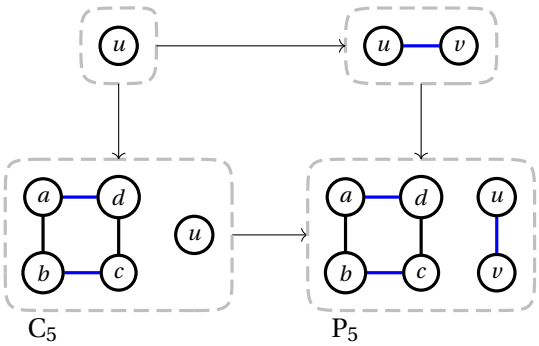
C_1

From this pushout, we can derive three epimorphisms, still assuming that the resulting morphisms from the extremities of the span are monomorphisms.



$T_{1,1}$ $T_{1,2}$ $T_{1,3}$

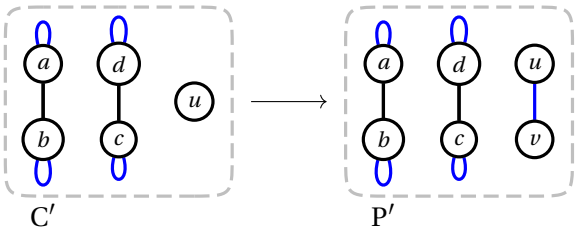
The resulting application condition is of the form $\bigwedge_{i\in I} \forall C_i, \bigvee_{k\in K} \exists T_{i,k}$. The C_i's are obtained from the first set of epimorphisms, and the $T_{i,k}$'s from the second, i.e., after the pushout from the C_i's. Restricted to C_1, we then obtain the application condition $\forall C_1, \exists T_{1,1} \vee \exists T_{1,2} \vee \exists T_{1,3}$. This condition is a tautology since C_1 is isomorphic to $T_{1,1}$. Similar tautologies will be found for the graphs C_2, C_3, and C_4. We obtain the following pushout for the graph C_5.



C_5 P_5

We then obtain an application condition of the form $\forall C_5, \bigvee_{k\in K} \exists T_{5,k}$ for a family of graphs $T_{5,k}$. Among them, one is isomorphic to the graph P_5 from the previous pushout. This right application condition cannot directly be discarded as it is not a tautology. Thus, the right application condition

is shifted into a left application condition. Formally, this shift is obtained by applying the reverse rule (i.e., the rule $R \overset{i_R}{\hookleftarrow} K \overset{i_L}{\hookrightarrow} L$ if the original rule was $L \overset{i_L}{\hookleftarrow} K \overset{i_R}{\hookrightarrow} R$) to the constructed graphs. We obtain an application condition of the form $\forall C', \bigvee_{k \in K} \exists T'_k$. Applying the reverse rules to C_5 and P_5 yields the graphs C' and P' as follows, where P' is one of the T'_k's.



Now, the constraint $\forall(u), \exists(u \text{—} v)$ implies $\forall C', \exists P'$ and thus $\forall C', \bigvee_{k \in K} \exists T'_k$. Note that the left-hand side of the rule does not satisfy $\forall(u), \exists(u \text{—} v)$ as the 2-arcs are loops. However, if we use boolean formulas to consider the possibility of loops, we would obtain a tautology.

The complete treatment of the incident arcs constraint would require accounting for oriented arcs and preventing a node from being the source of two arcs for a given dimension. Obtaining a complete preserving framework also supposes to take care of the non-orientation and cycle properties. The three topological constraints can be expressed using first-order formulas, as seen in Section 3.2, which are equivalent to nested conditions.

3.4.3 Nested application conditions

Application conditions have proven helpful in many cases but too restrictive in others. They were extended to nested application conditions (or nested conditions) in [89, 90, 92] and shown to be expressively equivalent to first-order graph formulas [91], fulfilling the requirements presented in [151]. In [58], standard results were proven for rules with nested conditions. In particular, Local-Church Rosser, Parallelism, Concurrency, and Amalgamation theorems hold for rules with nested conditions in $\mathscr{M}$-adhesive categories (a weaker form of adhesivity restricted to a subclass of monomorphisms, which we will present in Chapter 5).

Nested application conditions are defined on objects and (un)satisfied by morphisms from these objects. A nested condition on the initial object is called a constraint, and the uniqueness of morphisms from the initial object yields the satisfiability of constraints for objects. Like application conditions, nested conditions can be shifted over rules and morphisms, leading to the definition of constraint-preserving rules [91]. Applying such rules to a graph satisfying a constraint c always results in a graph that satisfies c.

Nested conditions have been used to verify graph programs with respect to a precondition and a postcondition, i.e., to demonstrate that the output satisfies a postcondition, provided the input satisfies a precondition. The first method is to construct a weakest precondition relative to the postcondition. The proof that the precondition implies the weakest precondition yields the correctness of the program [46]. The construction of weakest preconditions of a rule relative to a postcondition is discussed in [92] for nested application conditions. The beginning of the construction is similar to the discussion from Section 3.4.2. First, the postcondition's constraint is transformed into a right (nested) application condition. Then, the nested condition is shifted

from right to left. Lastly, the left (nested) application condition is transformed into a constraint modeling the weakest precondition (relative to the postcondition).

Once the weakest precondition is obtained, one still has to show that the precondition on the graph implies the weakest precondition of the rule relative to the postcondition. Since nested conditions are equivalent to first-order formulas on graphs, this verification can be delegated to a solver. Alternatively, Pennemann [140] developed a sound and complete (yet not guaranteed to terminate for unsatisfiable conditions) algorithm SEEKSAT for the satisfiability problem "$\exists G \in \mathscr{C}, G \models c$?" where $\mathscr{C}$ is a weak adhesive HLR category, and c is a nested condition. He also proposed a sound resolution-based calculus for proving nested conditions and developed a theorem prover PROCON based on this calculus. Both can be used to automatize the proof that the precondition implies the weakest precondition. As discussed in [50], the tools developed by Pennemann require potentially expensive computations. Thus, using another verification framework or a less expressive formalism may be more suitable depending on the aim of the study. For instance, the algorithm presented in [50] applies backward propagation to show that forbidden states cannot be reached from a safe state, assuming a finite number of transitions and forbidden states but a possibly infinite number of initial states. The second method is to provide a proof system and show its soundness with respect to the operational semantics [100]. In [142], the authors provide partial correctness of graph programs in the graph programming language GP [141] and show soundness with respect to the operational semantics of the language.

3.4.4 Consistency preservation at design time

From a theoretical point of view, nested application conditions sound like a suitable framework to define our consistency constraints and conditions to ensure the well-formedness of geometric objects. In particular, the constraints introduced in Section 3.2 can be defined with first-order graph formulas. However, we have different concerns about establishing consistency preservation.

We are not interested in proving a model specification for a graph transformation system. In such cases, users usually provide well-designed rules for which checking that formulas are tautologies can be efficiently done with a solver-based approach. If we were to define the three topological constraints with nested conditions, we would obtain tautologies for correct rules. However, our ambition is to provide a formal framework for the design of modeling operations. Our use cases are users that provide ill-formed rules and expect feedback on their design, which might typically fall into the non-terminating of Pennemann's algorithm. Besides the design of a geometric modeling tool that preserves the object's well-formedness, we have a second motivation for studying consistency preservation. We also seek to provide a debugger-like experience to the user. More precisely, we strive to provide real-time feedback while designing an operation. Such feedback should clarify which elements are missing or should be removed to ensure that the rule preserves the model's consistency. The computation of the weakest precondition relative to a postcondition also raises the question of the feasibility of the computation. As seen in Section 3.4.2, the computation is already consequent for a small rule with only a fraction of the incident arcs property. These consequent computations raise concerns for more significant rules when all constraints are considered, especially for design time verification.

Alternatively, we could consider any rule written by the user as valid and derive a left application condition from the constraint. Each operation designed by the user would be augmented

with an application condition. The transformation becomes subject to the satisfiability of the condition, following the standard approach to graph rewriting with application conditions, This solution approach raises two critical issues. First, the rule designer might not be the person using the modeling tool. Therefore, the tool user might get modeling operations that are not applicable in all cases considered similar from a geometric modeling point of view. In particular, in the case of poorly written rules, the tool user might get operations that are never applicable. Such a case is especially plausible if the rule designer did not receive any help writing the rules. We also have to consider that the rule designer might not be an expert in either the theory of graph rewriting or the specifics of the underlying model. Secondly, applying a rule with positive and negative application conditions requires checking the (non-)existence of a structure around the occurrence of the match. With nested application conditions, this verification essentially becomes recursive. In this sense, testing an application's condition can become a much harder matching problem. Geometric operations may modify objects with millions of elements resulting in computation-heavy processes. Hence, runtime checks should be reduced to a bare minimum, and providing static checks at design time appears to be the best solution.

Let us consider the example of the gluing condition. It states that a rule is applicable whenever the match and the left morphism of the rule admits a pushout complement. When rewriting graphs, the conditions can be reduced to the dangling condition that ensures no unmatched arc loses its source or target. In the framework of nested application conditions, the gluing condition can be defined as a condition on the left-hand side of the rule. Thus, when the condition is not fulfilled, the rule is inapplicable. In our framework, the gluing condition can be statically checked on the rule (Fact 1). Then, we are guaranteed that any match admits a pushout complement, and we no longer need to check the condition when applying the rule. Our approach is similar to the one developed in [19], where the authors tackle consistency preservation in the ECLIPSE Modeling Framework (EMF). They provide a formalization of EMF model transformations that preserve EMF consistency (containment relations) with typed attributed graph transformations. This formalization revolves around six ad hoc conditions that define allowed transformations. Our choice of design for consistency preservation can be related to the approach used in the chemical graph transformation MØD software [2]. In MØD, chemical molecules are modeled as graphs, while chemical reactions are specified as DPO rules. The chemical consistency of the molecule has to be satisfied. Otherwise, the graph does not model an existing chemical entity. For instance, rules should not create parallel edges. Such a property can be expressed with a NAC for every pair of vertices unconnected in the left-hand side but connected in the right-hand side. Rather than using NACs, the property is directly encoded in the algorithms.

Theorems 1, 2, and 3 express consistency preservation as simple syntactic conditions on the rule. Intuitively, such conditions are possible because of our specific framework, namely combinatorial graphs, which are highly regular. As a final remark, we want to highlight that Theorem 3 is stronger than the theorems presented in previous work [15, 14] for the preservation of the cycle constraint.

Summary of the chapter's contributions

In this chapter, we defined both Gmaps and Omaps as graphs via three topological constraints. These constraints (incident arcs, non-orientation, and cycle) express properties of the permutation system used in the combinatorial approach. They are defined locally on topological graphs, i.e., graphs labeled on arcs with dimensions. Modeling operations, i.e., modifications of the geometric object, are formalized with DPO rules. The rules are subject to conditions to ensure that a well-formed object is transformed into a well-formed object. One condition is derived from each topological constraint. Our conditions on rules implicitly exploit that rules are linear, allowing the comparison between the rule's left- and right-hand sides. Furthermore, the conditions for the non-orientation and cycle properties assume that rules are combinatorial, i.e., that they preserve the incident arcs constraint.

We defined conditions in a local approach meaning that it suffices to check some properties on the neighboring elements of a node to ensure the preservation of a constraint. In particular, all conditions can be checked statically on the rules via interaction over the set of nodes. Our approach accounts for a restricted framework imposed by the combinatorial graphs and rules but allows for efficient algorithms that ensure the preservation of the constraints. These algorithms will be presented in Chapter 4, along with rule schemes that generalize rules based on a topological pattern. The key idea is to analyze the rules at design time so that the rule designer can be assisted while writing an operation.

As a final note, the proof of Theorem 3 published in [139] is also the first published proof for the preservation of the cycle constraint.

Chapter 4

Generalizing operations via abstraction of the topology

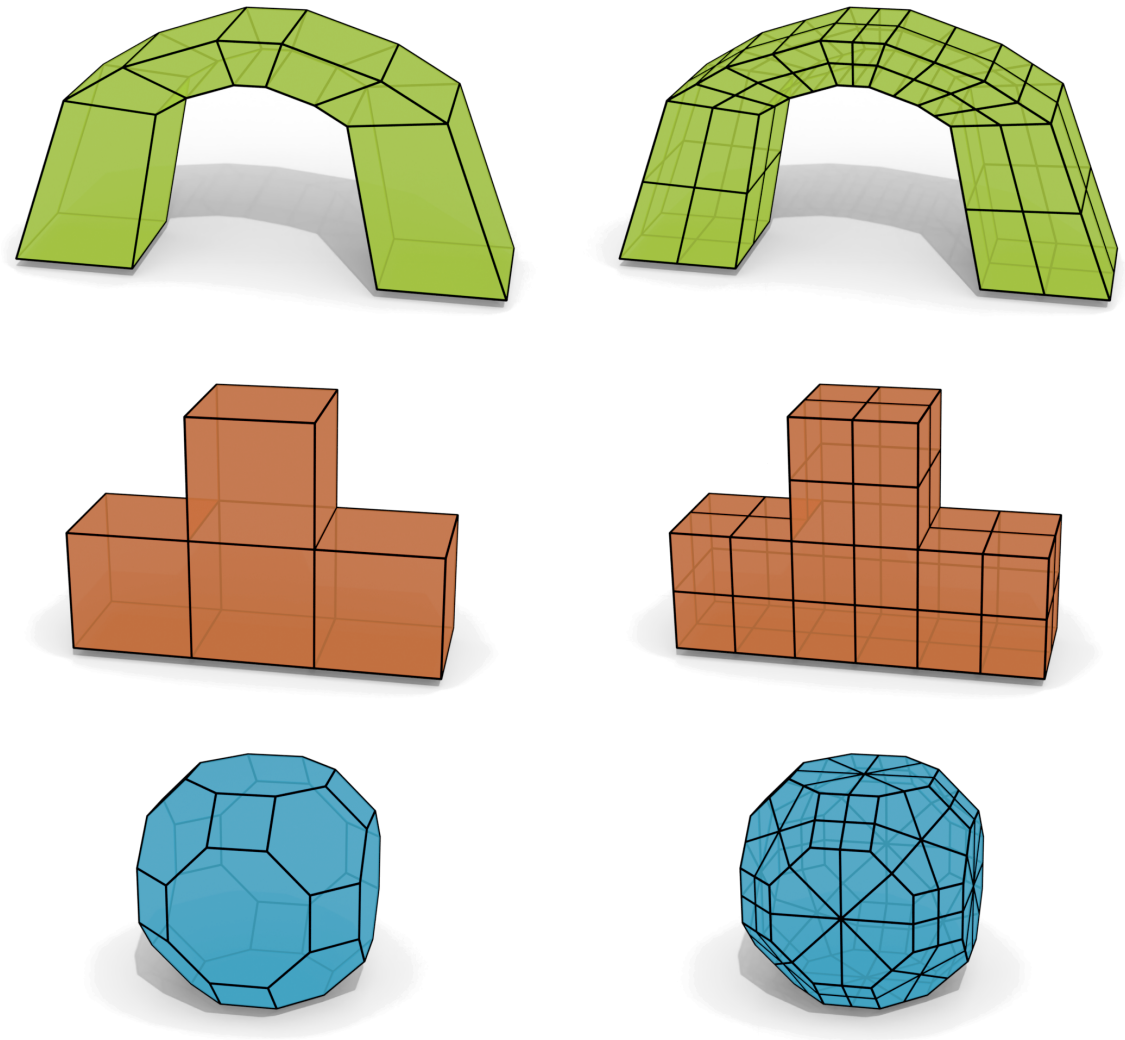


Figure 4.1: These three transformations (from left to right) correspond to distinct graph transformation rules, although they semantically describe the same modeling operation, namely the quad subdivision.

Personal note on the chapter

In this chapter, I investigate the generalization of DPO rules with a functorial, product-based approach. With such a generalization, I obtain a formal framework to reason about modeling operations. Some ideas developed in the second part of this manuscript exploit this formal framework. For a reader less proficient with category theory, which may be lost in some of the constructions and proofs of this chapter, we will present a fragment of this framework, explained in non-categorical terms, in Chapter 6.

Contents

The topological framework defined in Chapter 3 does not provide operations in the sense of geometric modeling. A direct derivation transforms the occurrence of a match in a graph. Therefore, we can modify a quad or a triangle by matching the corresponding (part of) Gmap or Omap. However, the transformation of this quad or triangle may have the same semantic description from the perspective of geometric modeling. For instance, we can triangulate a face. If the initial face is a quad, we obtain four new triangles, while an initial triangular face only yields three such triangles. These two transformations correspond to distinct graph rewriting rules but describe two instances of the same geometric modeling operation. Therefore, we need a generalization mechanism that allows the parameterization of operations. In previous works, e.g., [144], [16], or [12], the parameterization is achieved with variables in a setting similar to [101]. Here, we propose a category-based setting using a product to simulate a relabeling function while allowing the copying of the structure. The complete instantiation of the generalized rules can be formalized as a functor once the graph used to instantiate the rule has been chosen.

The chapter is organized as follows. First, we will review some insights into geometric modeling and how operations are typically designed in this domain (Section 4.1). In this review, we also discuss why the existing technics to extend graph transformations do not offer a satisfactory solution. Section 4.2 provides some intuition about using products to design graph transformations. We will use this product-based approach on combinatorial graphs to design abstract rewriting rules in Section 4.3. We will see that the incident arcs property, i.e., the use of permutations to represent geometric objects, is the cornerstone of the model. This property provides a rich framework to define interesting abstractions of graph transformations via rule schemes using products. Since the operations are performed using direct transformations from DPO rewriting, conditions on rules need to be lifted to rule schemes. Lifting the conditions from DPO rules to rule schemes is relatively straightforward for the incident arcs (Section 4.4) and non-orientation conditions (Section 4.5). However, the extension of the cycle condition for rules comes with additional constructions to deal with possible orientation for dimensions (Section 4.6). The last section (Section 4.7) wraps up the approach, providing results for the preservation of Gmaps and Omaps consistency. In particular, algorithms for analyzing rules are presented.

In this chapter, we will consider that all assumptions made in Chapters 2 and 3 hold. In particular, we consider linear rules described as partial monomorphisms and monic matching. Besides, we require that an arc between two nodes mapped by a partial mono is also mapped by the partial mono (Assumption 4 in Chapter 3). We use the notation $r = \mathrm{L} \rightarrowtail \mathrm{R}$ for a rule $r = \mathrm{L} \xhookleftarrow{i_{\mathrm{L}}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_{\mathrm{R}}} \mathrm{R}$ and mostly work with combinatorial graphs from a category $\mathbb{D}$-**CGraph**. All contributions of this chapter, namely Sections 4.3 to 4.7, have been published in [139].

4.1 Topological operations are more general than DPO rules

Given that Gmaps and Omaps are defined as graphs, we formalized the topological part of modeling operations as graph transformations (see Chapter 3). However, this formalization does not meet the standards to describe modeling operations. This section reviews these standards and discusses solutions to meet them.

The operation depicted in Figure 4.2 subdivides the face on the left to derive the four faces on the right. This example is a specific instance of the quad subdivision dedicated to refining the

topological structure of meshes [22]. This operation adds new vertices at the center of the face and the middle of each edge. An edge links the face center and the midpoint of the original edges.

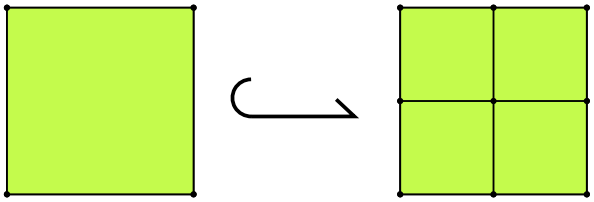
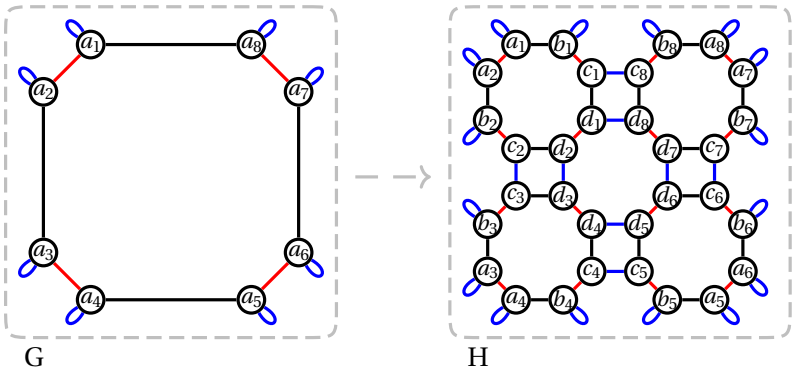


Figure 4.2: Quad subdivision of a face.

The cellular decomposition (see Chapter 3, Section 3.2.2) yields the operation depicted by the Gmaps of Figure 4.3, for the object of Figure 4.2. The initial square face is represented by 8 nodes that are preserved by the transformation. The four 1-arcs and 2-arcs are preserved while the 0-arcs are removed. The midpoints, the face center, and the linking edges are added. Based on the status of the element (added, preserved, and deleted), we can try to deduce possible rules for the operation.



G H

Figure 4.3: Gmap cellular decomposition of the face subdivided in Figure 4.2.

The rule in Figure 4.4 appears inconvenient because it does not specify that the matched element should be included in the same face. The rule in Figure 4.5 might seem the most general, but it does not match the whole edges of the initial face. However, this rule does not satisfy $\mathtt{C}_r(0,2)$, the cycle constraint for the dimensions 0 and 2. Indeed, the cycle constraint would be violated if this rule was applied to a non-isolated face. Finally, the rule in Figure 4.6 will never break the model consistency but is only applicable to isolated faces. Therefore, none of the presented rules offer an entirely satisfying solution. Besides, the rules presented here only apply to faces with four vertices, while the geometric modeling operation is applicable to any face regardless of the number of edges. For instance, we cannot obtain the subdivision of a triangular face illustrated in Figure 4.7.

Therefore, we need a generalization process. Such a process should be independent of the context and the underlying topology. For instance, the quad subdivision aims at subdividing surfaces for mesh refining. Each face of the mesh has to be subdivided. Since we are considering rewriting in the category of edge-labeled graphs, one could iteratively apply the operation on each face of the surface. Unfortunately, this approach would provide incorrect results as the subdivision of one face increased the number of edges on the surrounding faces. Details on the incorrect result are presented in Example 41. Similarly, one could try to apply the operation on every face simultaneously. This approach is not conceivable because of the concurrent modification of edges.

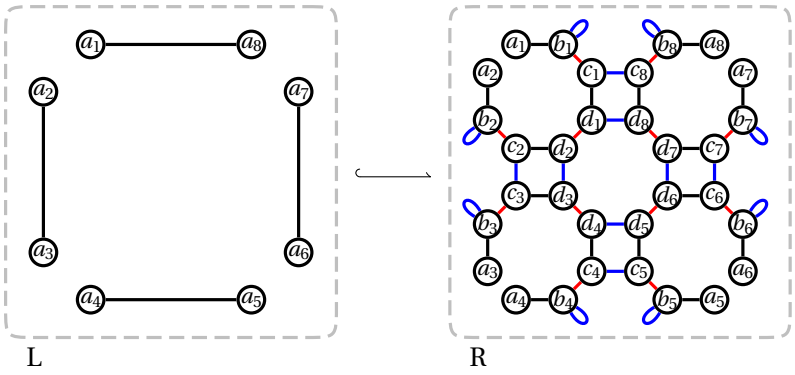


Figure 4.4: Minimal rule to obtain the transformation of Figure 4.3.

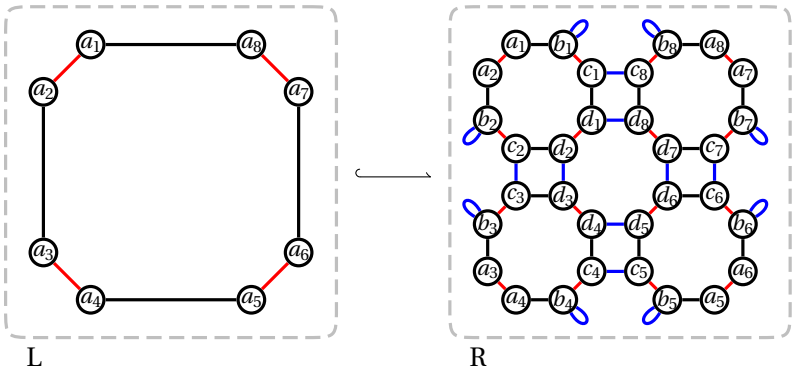


Figure 4.5: Intermediate rule to obtain the transformation of Figure 4.3.

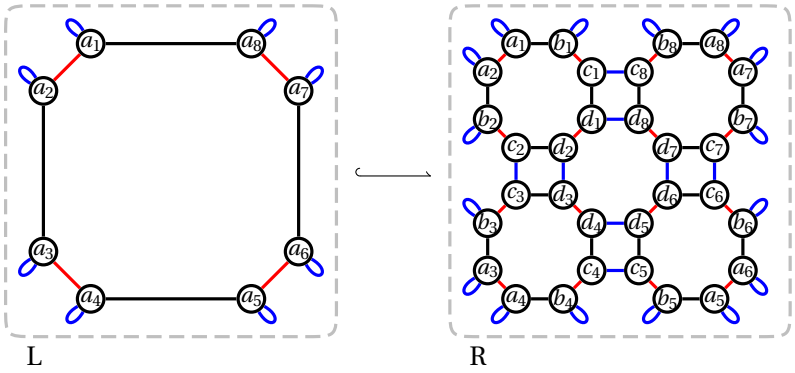


Figure 4.6: Maximal rule to obtain the transformation of Figure 4.3.

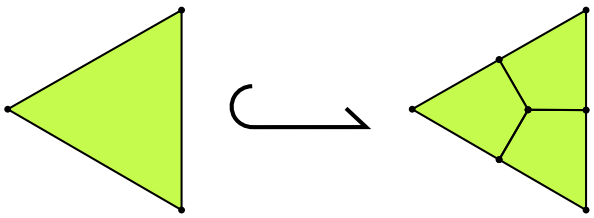
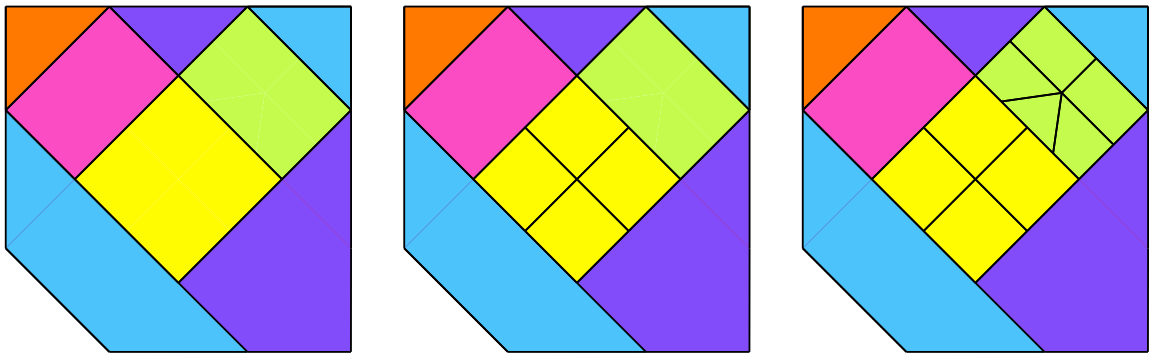


Figure 4.7: Quad subdivision of a triangular face.

Example 41 (Invalid sequential application of the quad subdivision)**.** A sequential application of the face subdivision application is shown in Figure 4.8. This sequence of operations is incorrect. Indeed, the edge between the yellow and green faces is split twice. First, the subdivision of the yellow face yields two edges (Figure 4.8b). Secondly, these two new edges are split again



(a) (b) (c)

Figure 4.8: Sequential application of the face subdivision operation: (a) initial object, (b) subdivision of the yellow face, and (c) subdivision of the green face.

when the green face is subdivided (Figure 4.8b). As a result, three vertices are added instead of one.

In the sense of graph transformations, the transformations are neither sequentially nor parallel independent (see [55] or [57, Chap.3]). Parallel-dependent transformations can be amalgamated into a transformation thanks to the amalgamation rule [20]. The computation of the amalgamation rule is done based on the transformations and principally used as a synchronization mechanism [174, 153]. Likewise, sequentially-dependent transformations can be reduced to a single transformation using the concurrency rule [64, 58].

One way to use these results would be to decompose a transformation into simpler transformations (and use the results to aggregate these simple transformations into more complex ones). For instance, one could have built a transformation for one node and then applied it to each node of the topological cell. Such a solution would fulfill the needs of a geometric modeler, i.e., specifying modeling operations that do not depend on the underlying topology of the object under modification. Nevertheless, this solution would be ill-suited as we need to link elements that belong to copies of different nodes from the topological cell. Besides, when considering Omaps, we need to link different copies of different nodes, which will lead us to introduce path-related abstraction with a product construction. Indeed, the amalgamation and concurrency theorems focus on graph transformation systems, although we are interested in providing a rule editor with compact and straightforward operations. We want to generalize the transformations to abstract a part of the object's topology.

Previous works [144, 16, 12] used topological variables to perform this generalization. These topological variables, expressed in the sense of [101], allowed for two-layered rules [14]. Such rules allowed matching an orbit, i.e., a subgraph induced by a set of dimensions (see Chapter 3). This subgraph would be copied, and the arcs would be relabeled. Arcs would be added between copies, meaning that each node of one copy of the subgraph was linked to its alter ego in another copy. From an algebraic perspective, orbits in oriented maps require composing the underlying permutations on darts. From a graph perspective, orbits in Omaps require considering paths in the underlying graph. Because of this single modification, the approach based on dimension relabeling does not provide a convenient solution. As illustrated in Figure 4.9a, the quad subdivision of a face in a Gmap setting can be decomposed into various copies of the initial face where the initial arcs are relabeled. The corresponding transformation is given in Figure 4.9b in an Omap setting.

In Figure 4.9, nodes are colored and named based on the copy of the initial subgraph (from the left-hand side).

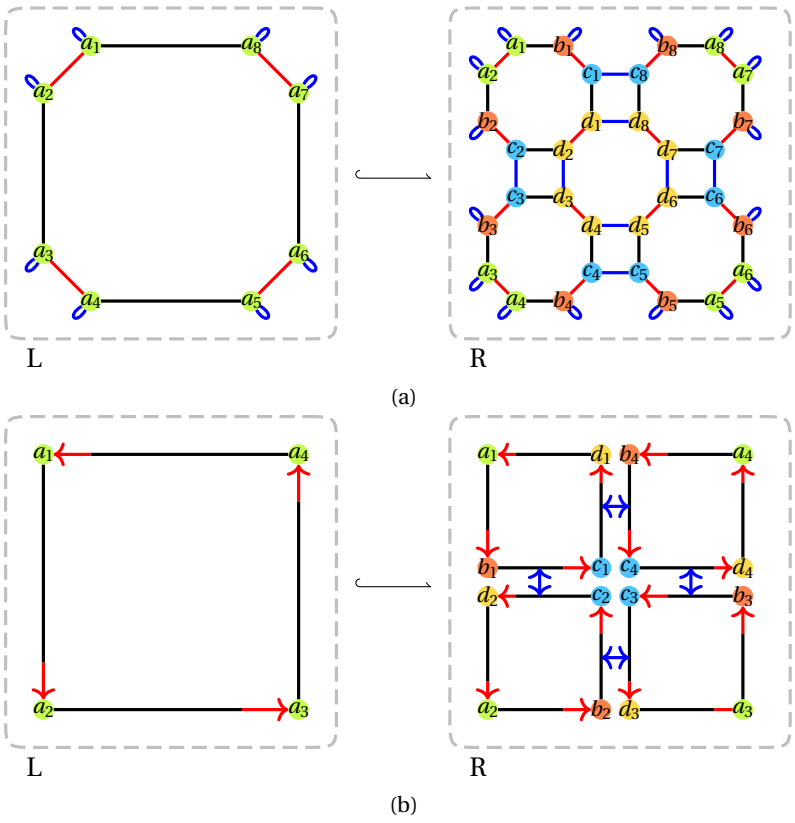


Figure 4.9: Generalization of the quad subdivision using copies and relabeling: (a) for Gmaps, and (b) for Omaps

For instance, in the right-hand side of the rule of Figure 4.9a, the nodes $c1$, $c2$, ... $c8$ belong to the same copy of the initial subgraph corresponding to the nodes $a1$, $a2$, ... $a8$. Similarly, nodes bi and di ($1 \leq i \leq 8$) describe two other copies of the initial subgraph. On the contrary, nodes $b8$, $c8$ and $d8$ belong to different copies. The number indicates the initial node of a copy node. As such, nodes $b8$, $c8$ and $d8$ are copies of the node $a8$. Rules with topological variables allowed arcs that either link nodes between a single copy, such as nodes $c1$ and $c8$, or link different copies of the same node, such as nodes $c8$ and $d8$. These topological variables meet the requirements for Gmaps. They will be extensively used in the second part of the dissertation as the crucial elements in the design of geometric modeling operations with Jerboa [12] (the Jerboa platform is presented in Chapter 6). Unfortunately, these topological variables are not straightforwardly extendable to Omaps. For instance, the rule for Omaps depicted in Figure 4.9b links node $b1$ to node $c2$, which cannot be expressed with topological variables. The regularity of the model still provides ways to describe the operation homogeneously. For instance, an attentive reader might notice that each node bi is linked with a 2-arc to the node $c((i \mod 4)+1)$. We will use words to encode paths describing these links.

We also want to highlight the size of graphs used in practice. In Figure 4.10a, the character consists of 12587 vertices, 12585 faces and a single connected component. The 3D character is represented by its boundary surface, i.e., a 2D manifold. The Gmap decomposition produces 100680 nodes and 302040 arcs in 2D, while the Omap decomposition yields 50340 nodes and 100680 arcs

(not drawn). We will see in Section 4.7 that the rule scheme for each model has a one-node left-hand side and a four-node right-hand side, regardless of the object size. The possibility to express operations as rules independent of the object size is an essential asset in geometric modeling, as the objects are usually huge. The quad subdivision depicted in Figures 4.10a and 4.10b has been realized with Jerboa with Gmaps (thus using topological variables). A zoom on the Gmap representation of the surface is provided in Figures 4.10c and 4.10d in the vicinity of the green face from Figure 4.10. Following the construction presented later in Chapter 5, the attribute values (here, the colors) are indicated on the nodes.

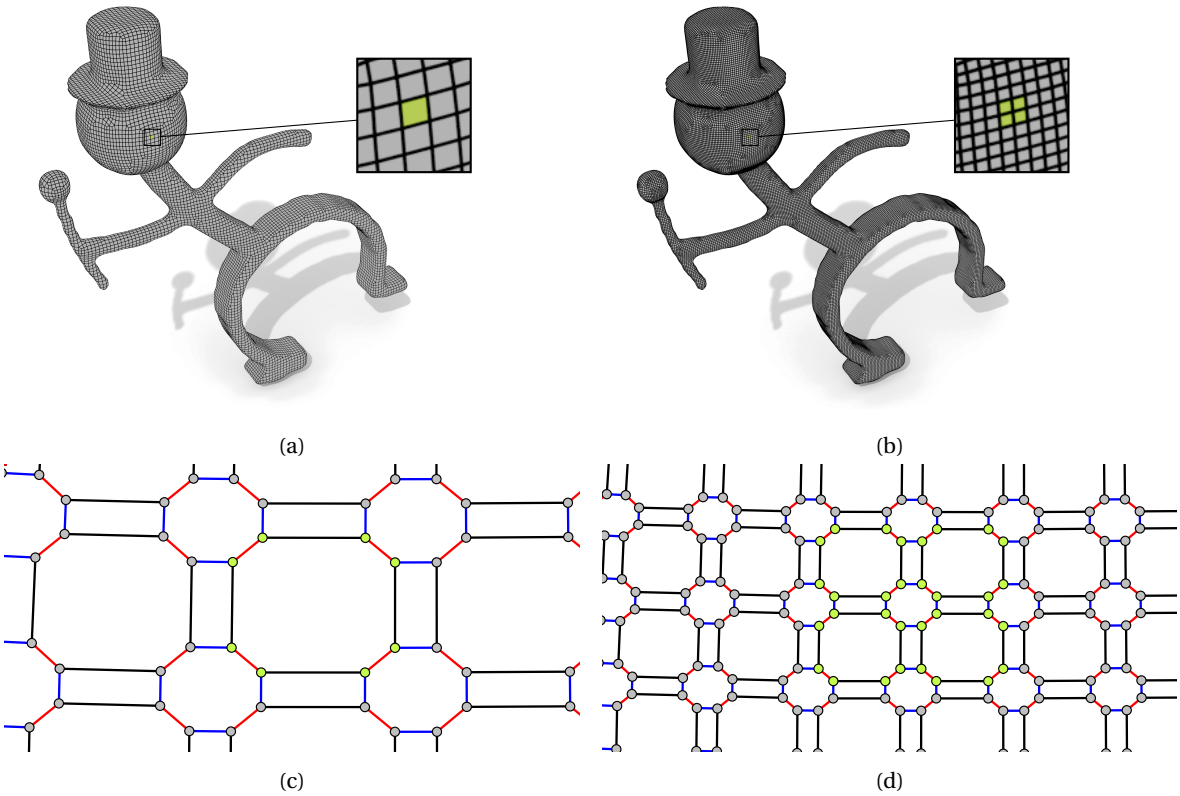


Figure 4.10: Quad subdivision of a surface: (a) surface before the subdivision, one face is colored in green to highlight the subdivision, (b) surface after the subdivision, (c) Gmap structure in the vicinity of the green face before the subdivision, and (c) Gmap structure after the subdivision.

4.2 Products to modify arc-labeled graphs

In this section, we discuss and motivate the use of products (the categorical one) as a solution to describe global operations on a graph, which we will use in this chapter to generalize the results of Chapter 3.

DPO rewriting allows modifying the labels of a graph. Typically, partially labeled graphs are considered, allowing to relabel nodes without matching all its incident arcs [93]. For arc relabeling, partially labeled graphs are not needed, and the relabeling of a i-arc into a j-arc can be achieved with a rule. This rule has two nodes in its left-hand side, right-hand side, and interface. The relabeling is described with a deleted i-arc in L, an added j-arc in R, and no arc in K. For instance the rule in the top span of Figure 4.11 relabels a ●-arc to a ●-arc. The occurrence of the match used for rule application is highlighted in green. It maps the node a to the node v, the node b to

the node x and the ●-arc between a and b to the ●-arc between v and x. The ●-arc is replaced by a ● one.

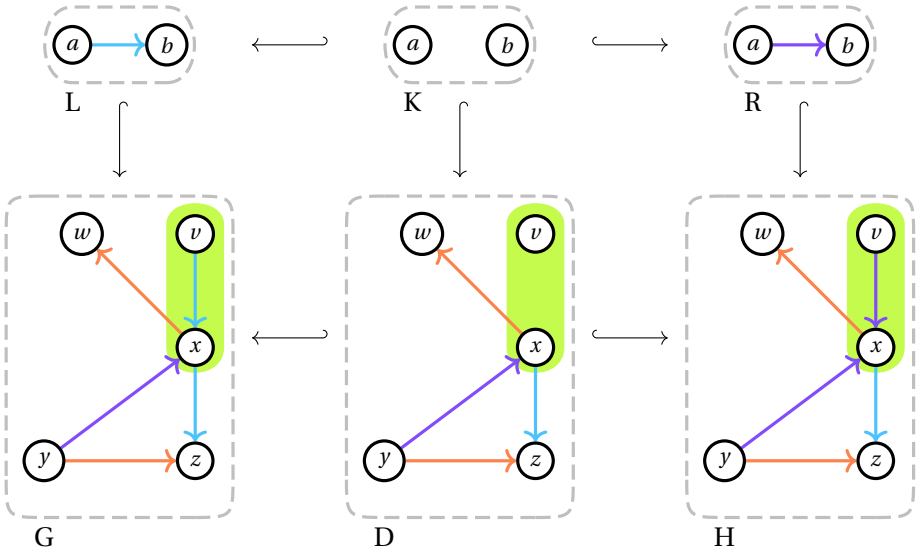


Figure 4.11: DPO transformation for relabeling a ●-arc to a ●-arc in {●, ●, ●}-**Graph**.

DPO rewriting models transformations based on structure deletion and creation, allowing for a straightforward definition of graph modifications in a preservative framework. However, DPO rewriting is ill-suited should the initial graph elements be cloned or specifications on their incident arcs be made. On the other hand, Node-Labeled-Controlled grammars [66] inherently allow copying, merging, or deleting parts of the graph. Bauderon provided a categorical definition of this approach [8] to simultaneously apply various modifications at different spots in a graph with a single mathematical operation, defined by a pullback.

Working in a category Σ-**Graph** where the arcs are labeled over an alphabet Σ, pullbacks over $\mathbb{1}_\Sigma$ are simply products (see Example 21 in Chapter 2). Indeed, $\mathbb{1}_\Sigma$, which has one node and one i-loop per letter in Σ is terminal in Σ-**Graph**. We now detail an example of a product in Σ-**Graph**. Recall that we might subscript the product with the alphabet for clarity purposes.

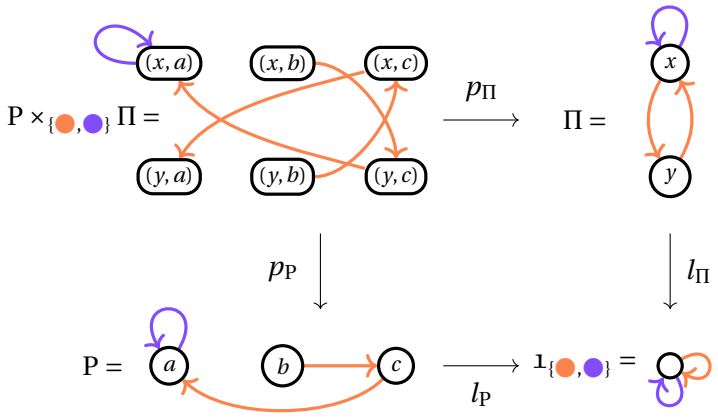


Figure 4.12: A product in {●, ●}-**Graph**.

Example 42 (Product in Σ-**Graph**)**.** Let P and Π be two graphs in {●, ●}-**Graph**, i.e., arc-labeled graphs over the alphabet Σ = {●, ●}. We can view the nodes of Π as different copies of the graph P. For instance, in the transformation of Figure 4.12 Π has two nodes x and y, meaning the nodes of P will be duplicated. The arcs of Π describe the transformations to operate on the arcs of P. Here, Π has three arcs. The ●-loop on x in Π and the ●-loop on a in P yield a ●-loop on (x, a) in $\mathrm{P} \times_{\{●,●\}} \Pi$. Similarly, the ●-arc from y to x in Π and the ●-arc from b to c in P produce a ●-arc from (y, b) to (x, c) in $\mathrm{P} \times_{\{●,●\}} \Pi$. Note that the ●-arc in P can not be associated with any arc of source y in Π. Thus, there is no ●-arc of source (y, a), (y, b) or (y, c) in $\mathrm{P} \times_{\{●,●\}} \Pi$.

In Chapter 3, we used DPO rewriting to transform the topological part of objects, represented as graphs. We extend this approach with a product-based generalization to express cloning and transformations on a global scale. This construction offers a simple yet sufficient construction for our invariant-preserving framework. This short section provides examples of transformations that can be realized with graph products.

4.2.1 Arc deletion

Let P and Π be two graphs in Σ-**Graph**. If Π consists of a single node, its loops describe operations to be carried out on the arcs of P. Thus, we can design the deletion of all arcs with a given label in a single operation. We can use a product construction to delete all the i-arcs for a label $i \in \Sigma$. This deletion is achieved as the product of P with $\Pi = \Delta_i$. Δ_i is the graph having only one node and $|\Sigma| - 1$ loops labeled d for every d in $\Sigma \setminus \{i\}$. Such an operation is illustrated in Figure 4.13 where all the ●-arcs of P are deleted.

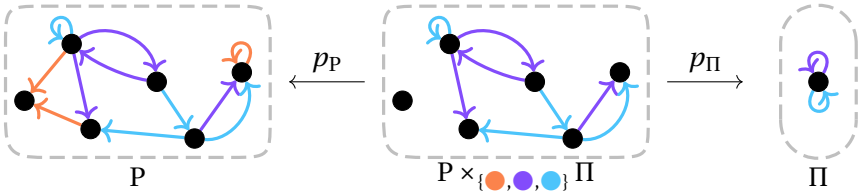


Figure 4.13: Deletion of all the ●-arcs using a product in {●, ●, ●}-**Graph**.

4.2.2 Arc relabeling

In the case of relabeling, P is the graph in which we want to relabel arcs, whereas Π stores the relabeling. Let us consider $\Sigma = \{i,\ i',\ j,\ j',\ k,\ d\}$ and assume we want to systematically relabel i into j, i' into j', keep k unchanged and delete d. These modifications are to be carried out on all arcs of the corresponding labels. This global operation can be encoded with a relabeling function defined by the set of pairs $\{(i, j), (i', j'), (k, k)\}$. More generally, a relabeling operation is defined by a set of pairs $\mathfrak{R} = \{(i, j) \mid i \in \Sigma, j \in \Sigma\}$ such that for any i in Σ, there is at most one j in Σ satisfying $(i, j) \in \mathfrak{R}$. Such a set will be called a *relabeling set*. A letter i in Σ such that (i, i) is in $\mathfrak{R}$ is a label left unchanged by the relabeling. Conversely, a letter i in Σ such that for all j in Σ, (i, j) is not in $\mathfrak{R}$ is a label deleted by the relabeling. As $\mathfrak{R}$ is a subset of Σ^2, the product is expressed in the category Σ^2-**Graph**. The Σ-graph P to be renamed must be converted into a labeled graph with labels in Σ^2. We will consider all the possible renamings. For each arc i, we will replace it with as many

arcs as possible labeled with pairs of the form (i, j) with j in Σ. Moving back and forth between the two categories is achieved with functors. The embedding[1] functor $\mathbb{E}_\Sigma \colon \Sigma$-**Graph** $\to \Sigma^2$-**Graph** transforms an i-arc ($i \in \Sigma$) into $|\Sigma|$ arcs labeled (i, j) for each $j \in \Sigma$. Considering all pairs (i, j) for $j \in \Sigma$ makes each relabeling possible. The projecting functor $\pi_\Sigma \colon \Sigma^2$-**Graph** $\to \Sigma$-**Graph** keeps the relabeled part of arc labels. Formally, the embedding and projecting functors are defined as follows.

Definition 41 (Embedding and projecting functors)**.** *Let us consider a finite alphabet* Σ.

The embedding functor $\mathbb{E}_\Sigma \colon \Sigma$-**Graph** $\to \Sigma^2$-**Graph** *transforms:*

- *A graph* $G = (V, E_G, s_G, t_G, l_G)$ *into a graph* $H = (V, E_H, s_H, t_H, l_H)$ *such that for every* i*-arc* e *of* E_G *with* i *in* Σ*, for all* $j \in \Sigma$*, there exists an* (i, j)*-arc in* E_H *with the same source and target, written* $e^{(j)}$.
- *A morphism* $m = (m_V, m_E) \colon A \to B$ *into a morphism* $m' = (m_V, m'_E) \colon C \to D$ *such that for every* e_A *in* E_A*, for every* j *in* Σ*,* $m'_E(e_A^{(j)})$ *is the arc* $m_E(e_A)^{(j)}$.

The projecting functor $\pi_\Sigma \colon \Sigma^2$-**Graph** $\to \Sigma$-**Graph** *transforms:*

- *A graph* $G = (V, E, s, t, l_G)$ *into a graph* $H = (V, E, s, t, l_H)$ *such that* l_H *only keeps the second part of the label from* l_G.
- *A morphism* $m = (m_V, m_E) \colon A \to B$ *into a morphism* $m' = (m_V, m'_E) \colon C \to D$ *such that if an* (i, j)*-arc* e_A *is sent to an* (i, j)*-arc* e_B *by* m_E *then the* j*-arc* e_C *built on* e_A *is sent to the* j*-arc* e_D *built on* e_B.

Let us consider the relabeling defined by relabeling set $\mathfrak{R}$. The construction works as follows :

1. Embed P in the category Σ^2-**Graph** to get $\mathbb{E}_\Sigma(P)$.
2. Construct the product $\mathbb{E}_\Sigma(P) \times_{\Sigma^2} \Pi_\mathfrak{R}$ in Σ^2-**Graph**, where $\Pi_\mathfrak{R}$ is the relabeling graph having only one node, and a loop labeled (i, j) for each pair (i, j) in the relabeling set $\mathfrak{R}$.
3. Apply the projecting functor π_Σ to $\mathbb{E}_\Sigma(P) \times_{\Sigma^2} \Pi_\mathfrak{R}$.

Example 43 (Relabeling with products)**.**

We consider relabeling in {●, ●, ●}-**Graph**. We consider the relabeling of all the ●-arcs of a graph into ●-arcs, while deleting the initial ●-arcs, as shown in Figure 4.14. First, we apply the embedding functor $\mathbb{E}_{\{●,●,●\}}$ from {●, ●, ●}-**Graph** to {●, ●, ●}^{2}-**Graph**. From each arc, three arcs are built, having ●, ● and ● as second label. The arcs are labeled with pairs of colors in {●, ●, ●}^{2}-**Graph**. Thus, we represent an arc with two arrows, such as ●⟶⟼●. The first one is close to the source and colored with the first value of the pair, in this case, ●. The second arrow, close to the target, is colored with the second label, here ●. Similarly, the bicolored loop in Π is a (●, ●)-loop. Then, we construct the product with the relabeling graph (written Π in the figure). The relabeling set is $\mathfrak{R} = \{(●, ●), (●, ●)\}$. This relabeling set yields the graph that consists of a single node and two arcs, each specifying the relabeling of one letter: ● is relabeled ●, ● is deleted and ● is left unchanged. The product construction only keeps the arcs that match

[1] The term 'embedding' in this chapter refers to the embedding of a category Σ-**Graph** into a category Σ'-**Graph** and should not be confused with the embedding of the topological structure into a geometric space studied in Chapter 5.

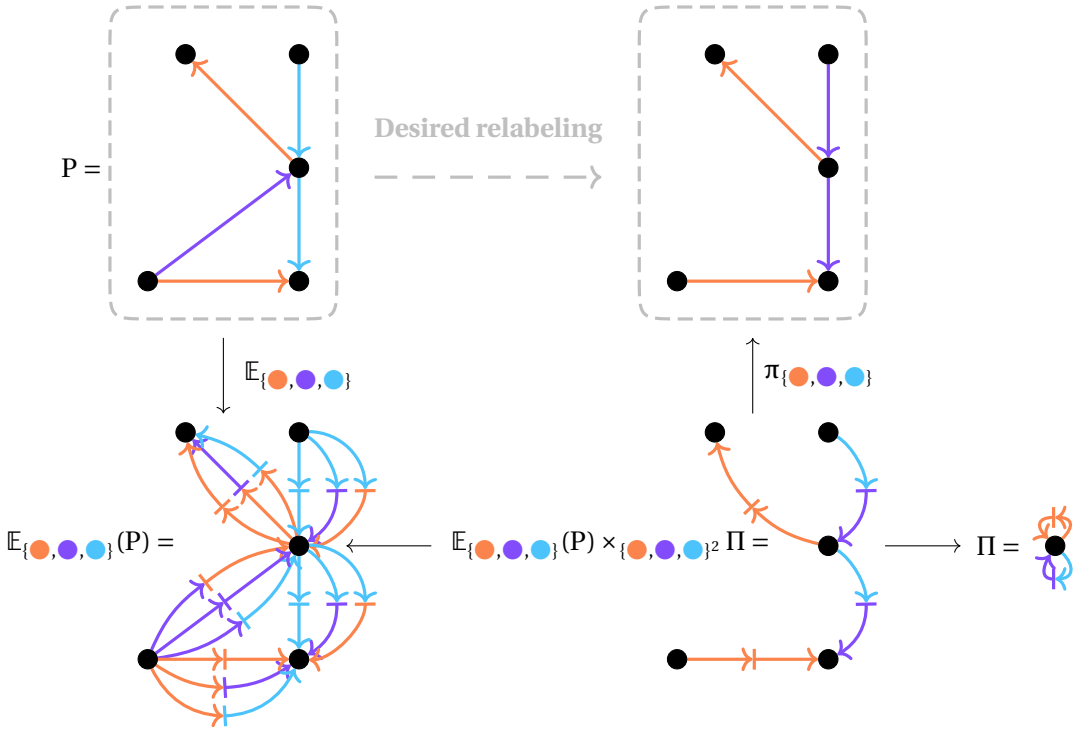


Figure 4.14: Global relabeling with a product.

the relabeling. Lastly, the projecting functor $\pi_{\{\bullet,\bullet,\bullet\}}$ erases the first label on the arcs, and we obtain the desired relabeling.

The graph transformations presented in this section exploit products with one-node graphs. In this chapter, we will use multiple node graphs to allow copying. Loops on the graph describe modifications on the copies, while non-arc loops stitch the copies together.

4.3 Abtract rules and their instantiation

As we have seen in Section 4.2, products offer a simple way to apply transformations, such as arc deletion or arc relabeling, on a global scale. Taking benefit from our specific graphs, we will use the product as a mechanism to generalize transformations of $\mathbb{D}$-combinatorial graphs in the context of the DPO approach.

Our product-based generalization of DPO rewriting will allow us to define a large family of operations on combinatorial graphs, taking advantage of their regularity with respect to the dimensions of $\mathbb{D}$. In Section 4.3.1, we introduce pattern graphs. Pattern graphs carry words of dimensions as labels on their arcs and enable us to select subgraphs to be modified in the combinatorial graphs. In Section 4.3.2, we define graph schemes as a way to encompass a transformation applied globally on a graph. We call instantiation of a graph scheme the operation (relying on a graph product) that transforms a pattern graph into a topological graph, based on a graph scheme. We define rule schemes as spans in the category of graph schemes in Section 4.3.3. Similarly, we can instantiate a rule scheme by instantiating its three graph schemes with the same pattern graph. Since we can instantiate a rule scheme with any pattern graph (assuming the right set of labels), rule schemes are rule abstractions, encoding infinitely many possible rewriting. Finally, in Section 4.3.4, we will discuss the mechanisms involved in the application of rule schemes, i.e., the

matching process.

4.3.1 Global extraction of paths

In a $\mathbb{D}$-combinatorial graph, every node is the source of a unique arc for each dimension in $\mathbb{D}$. Therefore, given a word w on $\mathbb{D}$, every node is the source of a unique w-path. In the relabeling operation defined in Section 4.2 we used pairs of the form (i, j) to rename a label i into j. Here, we will consider, more generally, pairs of the form (w, i) to create an i-labeled arc between the source and the target of a w-path.

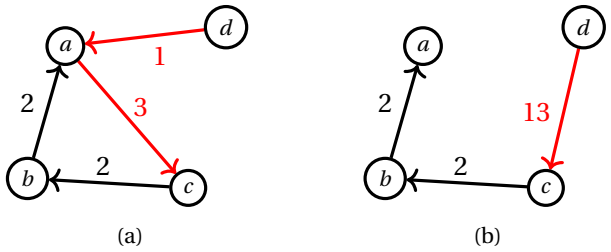


Figure 4.15: Representing paths with a graph: (a) a (topological) (1..3)-graph, and (b) a (pattern) $\{2, 13\}$-graph.

Therefore, the set $\mathbb{D}$ will also be used as an alphabet for more structured sets. We will consider categories $\mathbb{W}$-**Graph** where $\mathbb{W}$ is a finite subset of $\mathbb{D}^*$. Intuitively, arc labels in such graphs describe paths in an underlying graph. For instance, in Figure 4.15b, the red arc labeled 13 can be interpreted as the 13-path in the graph of Figure 4.15a. We will call pattern graph any graph labeled with words from $\mathbb{W}$.

Definition 42 (Pattern graph)**.** *Let $\mathbb{D}$ be a finite set of integers, and let $\mathbb{W}$ be a finite subset of $\mathbb{D}^*$. A $\mathbb{W}$-*pattern graph *is a graph from the category $\mathbb{W}$-***Graph***.*

Although the definition is general, we will extract pattern graphs from the Gmap or the Omap to be modified. The extraction of pattern graphs is postponed to Section 4.3.4.

The uniqueness of the source and target per dimension makes it possible to build paths by following arcs forward or backward. Thus, for a dimension i, we introduce the notation $\overline{i}$ to indicate that, from a given node v, the next node in the path is the unique node v', source of the i-arc whose target is v. We accordingly extend the labeling alphabet of the graphs.

Definition 43 (Conjugate dimensions)**.** *For any $d \in \mathbb{D}$, we define the* conjugate dimension $\overline{d}$ *which yields the* conjugate alphabet $\overline{\mathbb{D}} = \mathbb{D} \cup \{\overline{d} \mid d \in \mathbb{D}\}$.

The conjugation is extended to words on $\mathbb{D}^*$ such that the conjugate of a word $w = w_{(1)}\, w_{(2)} \dots w_{(n)}$ is the word $\overline{w} = \overline{w_{(1)}\, w_{(2)} \dots w_{(n)}} = \overline{w_{(n)}} \dots \overline{w_{(2)}}\, \overline{w_{(1)}}$.

Finally, the conjugation is extended to $\overline{\mathbb{D}}$ and $\overline{\mathbb{D}}^*$ by considering that for any dimension $d \in \mathbb{D}$, $\overline{\overline{d}} = d$.

Conjugation allows us to broaden the definition of a path in the case of $\mathbb{D}$-combinatorial graphs. For a $\mathbb{D}$-combinatorial graph $G = (V, E, s, t, l)$, we introduce the conjugate $\overline{\mathbb{D}}$-combinatorial graph $\overline{G} = (V, E \cup \overline{E}, \overline{s}, \overline{t}, \overline{l})$, such that:

- $\overline{E} = \{\overline{e} \mid e \in E\}$,

- functions $\overline{s}$, $\overline{t}$ and $\overline{l}$ on arcs in E are the corresponding functions of G, i.e., for $e \in E$, $\overline{s}(e) = s(e)$, $\overline{t}(e) = t(e)$ and $\overline{l}(e) = l(e)$,
- functions $\overline{s}$, $\overline{t}$ and $\overline{l}$ on arcs in $\overline{E}$ rely on the inversion of the underlying arcs, i.e., for $e \in E$, $\overline{s}(\overline{e}) = t(e)$, $\overline{t}(\overline{e}) = s(e)$ and $\overline{l}(\overline{e}) = \overline{l(e)}$.

From now on, for w in $\overline{\mathbb{D}}^*$, w-paths of a $\mathbb{D}$-combinatorial graph will implicitly refer to the w-paths defined on its conjugate version. In short, such w-paths are well-defined, and their target always exists because a node in G is the source and target of a unique i-arc for each dimension i in $\mathbb{D}$.

Fact 2. *The incident arcs constraint of a $\mathbb{D}$-combinatorial graph guarantees the existence and uniqueness (up to the starting node) of paths labeled by a sequence of conjugate dimensions.*

In order to build rule schemes as an abstraction of rules, labels on the conjugate alphabet will be used to encode the reconnection between nodes in the graph that supports the instantiation. We extend the definition of pattern graphs to graphs labeled over subsets of $\overline{\mathbb{D}}^*$. Figure 4.16 provides an example of a $\{\epsilon, 21, \overline{21}\}$-pattern graph. In this pattern graph, each node a, b, c, and d is the sources of a ϵ-arc, a 21-arc, and a $\overline{21}$-arc.

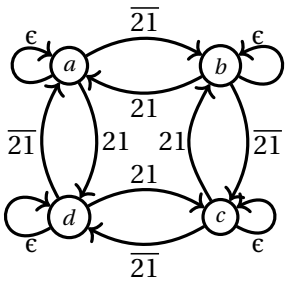


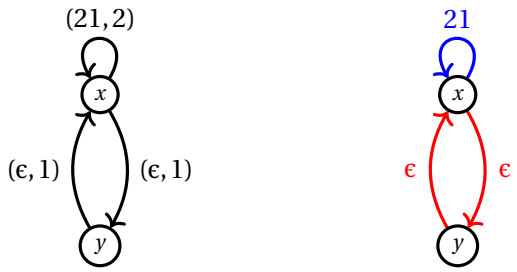
Figure 4.16: A $\{\epsilon, 21, \overline{21}\}$-pattern graph.

Intuitively, labeling arcs with paths offers greater freedom to characterize modified parts of the graphs and create arcs from such complex paths. Generally, any pattern graph can be used to instantiate a rule scheme. In practice, pattern graphs will be extracted from the graph modified by the rule scheme. We will discuss in Section 4.3.4 how to derive pattern graphs from a combinatorial graph. In the sequel, $\mathbb{W}$ is a finite set of words of $\overline{\mathbb{D}}^*$.

4.3.2 Graph schemes

Following the idea given in Section 4.2 of representing a graph modification with a product, we use a graph to represent the modification carried out on the pattern graph. Since pattern graphs belong to the category $\mathbb{W}$-**Graph** and we are trying to build objects from the category $\mathbb{D}$-**Graph**, the modification will rely on a function from $\mathbb{W}$ to $\mathbb{D}$. Therefore, we need graphs labeled with pairs from $\mathbb{W} \times \mathbb{D}$, which we call graph schemes.

Definition 44 (Graph scheme)**.** *Let $\mathbb{D}$ be a finite set of integers, and let $\mathbb{W}$ be a finite subset of $\mathbb{D}^*$. A $(\mathbb{W} \times \mathbb{D})$*-graph scheme *is a graph from the category $(\mathbb{W} \times \mathbb{D})$-***Graph**.



(a) (b)

Figure 4.17: A graph scheme: (a) a $\{\epsilon, 21\} \times \{1, 2\}$-graph scheme and (b) compact representation where only the $\mathbb{W}$ part of the label is written.

Example 44 (Graph scheme)**.** The graph of Figure 4.17a is a $\{\epsilon, 21\} \times \{1, 2\}$-graph scheme. Its arcs encode the functions $\epsilon \mapsto 1$ and $21 \mapsto 2$. To avoid overloading the figures, we will display graph schemes as in Figure 4.17b. The arc color indicates the dimension, i.e., the second part of the label, while the written word indicates the first part of the label.

Transformations now occur in the category of graph schemes. The construction is similar to the relabeling operation, with the functors $\mathbb{E}_\Sigma$ and π_Σ (Definition 41) replaced by:

- An embedding functor $\mathbb{E}_\mathbb{D}$: $\mathbb{W}$-**Graph** $\to$ $(\mathbb{W} \times \mathbb{D})$-**Graph** that transforms the arcs labeled by a word w in $\mathbb{W}$ into a set of $|\mathbb{D}|$ arcs labeled (w, d), for all $d \in \mathbb{D}$.
- A projecting functor $\pi_\mathbb{D}$: $(\mathbb{W} \times \mathbb{D})$-**Graph** $\to$ $\mathbb{D}$-**Graph** that only keeps the second part of the arc labels.

The projecting functor $\pi_\mathbb{D}$ essentially provides a way to obtain a topological graph from a graph scheme. We call core of a graph scheme its associated $\mathbb{D}$-graph.

Definition 45 (Core of a graph scheme)**.** *The* core *of a* $(\mathbb{W} \times \mathbb{D})$*-graph scheme* G *is its projection in the category* $\mathbb{D}$-**Graph** *via the projecting functor, i.e., the* $\mathbb{D}$*-topological graph* $\pi_\mathbb{D}(G)$.

To ease certain proofs and constructions, we will also consider the projecting functor $\pi_\mathbb{W}$: $(\mathbb{W} \times \mathbb{D})$-**Graph** $\to$ $\mathbb{W}$-**Graph** that only keeps the first part of the arc labels. The *instantiation* $\iota(\Pi, P)$ of a $(\mathbb{W} \times \mathbb{D})$-graph scheme Π on a $\mathbb{W}$-pattern graph P is similar to the relabeling operation given in Section 4.2. The instantiation is achieved as follows:

1. Embed P in the category of $(\mathbb{W} \times \mathbb{D})$-graphs to get $\mathbb{E}_\mathbb{D}(P)$.
2. Construct the product $\mathbb{E}_\mathbb{D}(P) \times \Pi$.
3. Apply the projecting functor $\pi_\mathbb{D}$ to $\mathbb{E}_\mathbb{D}(P) \times \Pi$ and get $\iota(\Pi, P)$, the instantiation of Π on P:

$$\iota(\Pi, P) = \pi_\mathbb{D}(\mathbb{E}_\mathbb{D}(P) \times \Pi).$$

This construction is summarized by the following commutative diagram, where the square is

a pullback, and double arrows represent functors.

$$\begin{array}{ccccc} \iota(\Pi,\mathrm{P}) & \xleftarrow{\pi_{\mathbb{D}}} & \mathbb{E}_{\mathbb{D}}(\mathrm{P})\times\Pi & \xrightarrow{p_{\Pi}} & \Pi \\ & & \downarrow{\scriptstyle p_{\mathbb{E}_{\mathbb{D}}(\mathrm{P})}} & & \downarrow{\scriptstyle !_{\Pi}} \\ \mathrm{P} & \xrightarrow[\mathbb{E}_{\mathbb{D}}]{} & \mathbb{E}_{\mathbb{D}}(\mathrm{P}) & \xrightarrow[!_{\mathbb{E}_{\mathbb{D}}(\mathrm{P})}]{} & \mathbb{1}_{\mathbb{W}\times\mathbb{D}} \end{array} \tag{4.1}$$

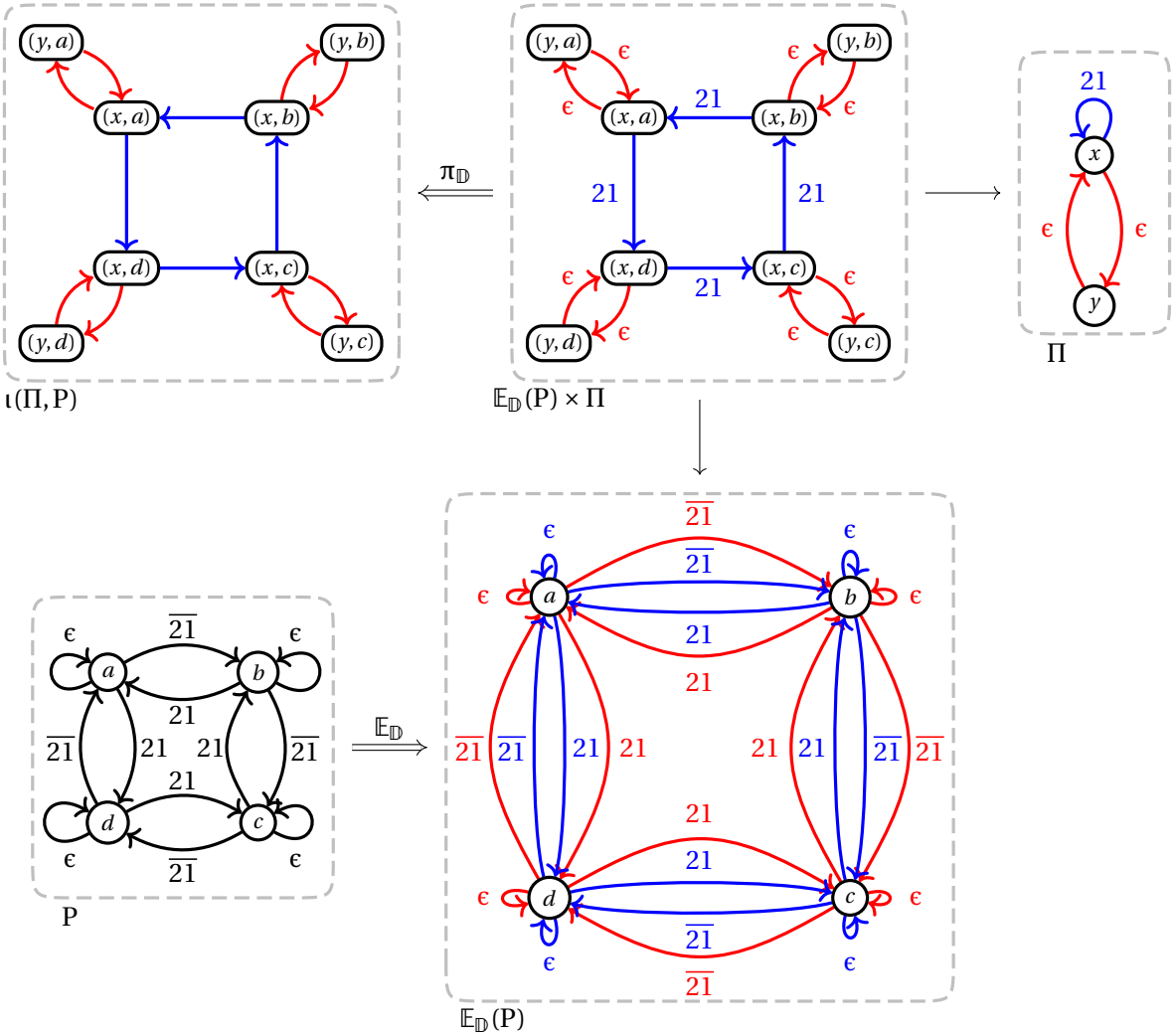


Figure 4.18: Instantiation of a graph scheme.

Example 45 (Instanciation of a graph scheme)**.** Let us consider the pattern graph from Figure 4.16 and the graph scheme from Figure 4.17. As depicted in Figure 4.18, the pattern graph is first embedded in the category $(\{21,\overline{21}\,\epsilon\}\times\{1,2\})$-**Graph** to give the graph $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$. The product $\mathbb{E}_{\mathbb{D}}(\mathrm{P})\times_{\Pi}$ between the embedded pattern graph and the graph scheme is then computed. Applying the projecting functor erases the first part of the labels and yields $\iota(\Pi,\mathrm{P}) = \pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P})\times_{\Pi})$. Figure 4.18 is a concrete example of the construction described in the diagram 4.1.

Given a fixed $\mathbb{W}$-pattern graph P, the instantiation on P defines a functor $\iota(-,\mathrm{P})\colon(\mathbb{W}\times\mathbb{D})$-**Graph** $\to \mathbb{D}$-**Graph** called the *instantiation functor*. Indeed, the categorical product can be defined as a functor, and we are simply composing functors.

4.3.3 Rule schemes

Having defined graph schemes, we can consider rule schemes as spans in the category of graph schemes. By application of the projecting functor, we also obtain the core of a rule scheme as a topological rule.

Definition 46 (Rule scheme)**.** *A* $(\mathbb{W}\times\mathbb{D})$-rule scheme $\mathscr{S} = \mathrm{L} \hookleftarrow \mathrm{R}$*, is a rule in the category of* $(\mathbb{W}\times\mathbb{D})$*-graph schemes. The* core *of the rule scheme* $\mathscr{S}$ *is the projection of* $\mathscr{S}$ *in the category* $\mathbb{D}$**-Graph***:*

$$\pi_{\mathbb{D}}(\mathscr{S} = \mathrm{L} \xhookleftarrow{i_{\mathrm{L}}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_{\mathrm{R}}} \mathrm{R}) = \pi_{\mathbb{D}}(\mathrm{L}) \xhookleftarrow{\pi_{\mathbb{D}}(i_{\mathrm{L}})} \pi_{\mathbb{D}}(\mathrm{L}\cap\mathrm{R}) \xhookrightarrow{\pi_{\mathbb{D}}(i_{\mathrm{R}})} \pi_{\mathbb{D}}(\mathrm{R}).$$

Since $\pi_{\mathbb{D}}(\mathrm{L})\cap\pi_{\mathbb{D}}(\mathrm{R}) = \pi_{\mathbb{D}}(\mathrm{L}\cap\mathrm{R})$, we can directly write $\pi_{\mathbb{D}}(\mathscr{S}) = \pi_{\mathbb{D}}(\mathrm{L}) \hookrightarrow \pi_{\mathbb{D}}(\mathrm{R})$.

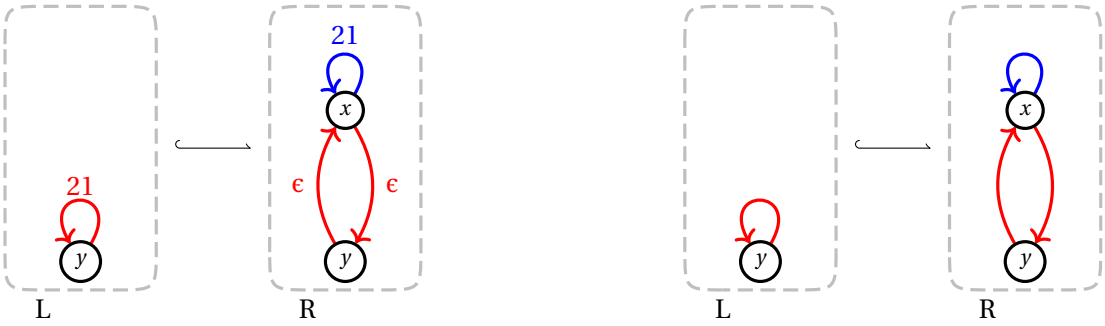


Figure 4.19: A $(\{21,\epsilon\},\{1,2\})$-rule and its core.

Example 46 (Rule scheme)**.** Reusing the graph scheme of Figure 4.17 as the right-hand side of a rule, we obtain the rule scheme of Figure 4.19a. Its core is given in Figure 4.19b and obtained by deleting the labels part on $\mathbb{W}$, i.e., only keeping the dimension.

Rule schemes are spans in a category of graph schemes. Although they could be used to transform graph schemes, our goal is to modify topological graphs. Therefore, we will instantiate the rule scheme by instantiating each of its graph schemes on the same pattern graph.

Definition 47 (Rule scheme instantiation)**.** *Let* $\mathscr{S} = \mathrm{L} \hookleftarrow \mathrm{R}$ *be a* $(\mathbb{W}\times\mathbb{D})$*-rule scheme and* P *a* $\mathbb{W}$*-pattern graph. The* scheme instantiation $\iota(\mathscr{S},\mathrm{P})$ *of* $\mathscr{S}$ *on* P *is the rule* $\iota(\mathrm{L},\mathrm{P}) \hookrightarrow \iota(\mathrm{R},\mathrm{P})$.

Similar to the construction of the core of a rule scheme, its instantiation is essentially the application of the $\iota(-,\mathrm{P})$ functor to the span for a fixed pattern graph P.

Example 47 (Rule scheme instantiation)**.** Figure 4.20 shows the instantiation of the rule scheme from Figure 4.19a with the pattern graph of Figure 4.16. Note that the instantiation of the graph scheme in Figure 4.18 is the instantiation of the rule's right-hand side and that we do not detail the instantiation of the left-hand side.

We now explain how the instantiation of a rule scheme can be used to obtain a rule that is applicable to a topological graph.

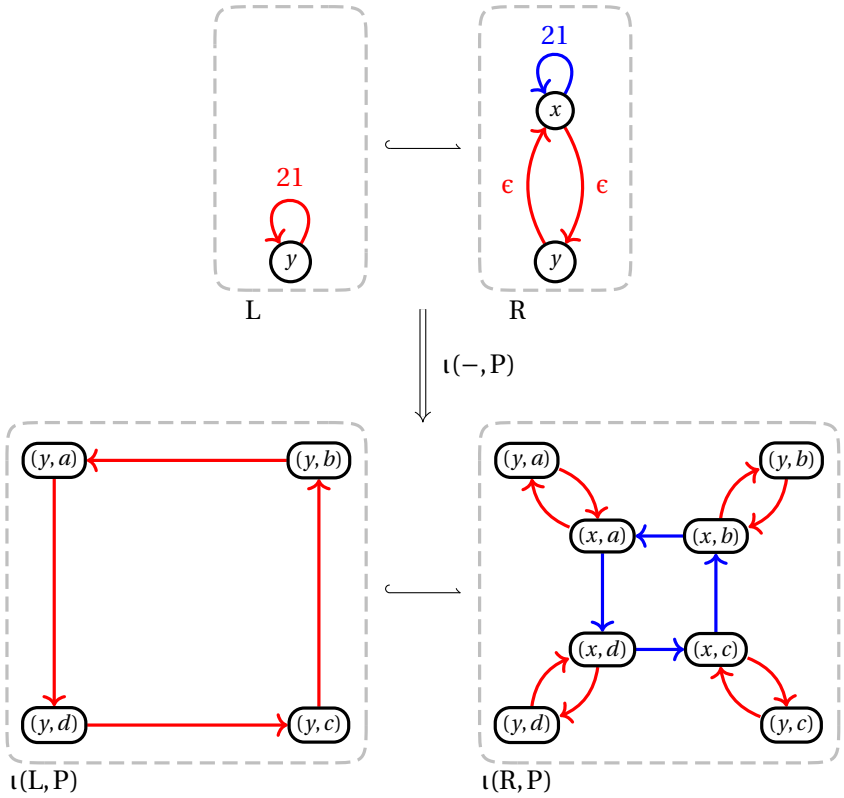


Figure 4.20: Instantiation of a rule scheme.

4.3.4 Application of rule schemes

The application of a rule scheme $\mathscr{S}$ to a $\mathbb{D}$-graph G is accomplished in a series of steps. First, a pattern graph $P = (V_P, E_P, s_P, t_P, l_P)$ for $\mathscr{S}$ is built from G. This construction essentially relies on the replacement of G by its $\mathbb{W}$-transitive closure. The idea is to replace any path $v \overset{w}{\rightsquigarrow} v'$ in G, where w is a word of $\mathbb{W}$, by an arc $e \in E_P$ such that $s_P(e) = v$, $t_P(e) = v'$ and $l_P(e) = w$. This construction has meaning only if the paths can be considered unambiguously. Therefore, we will restrict the construction to the category of combinatorial graphs $\mathbb{D}$-**CGraph**. Formally, the construction is achieved using a functor.

Definition 48 (Pattern functor)**.** *Let $\mathbb{D}$ be a finite set of dimensions and $\mathbb{W}$ a finite set of words in $\overline{\mathbb{D}}^*$. The* $(\mathbb{D}, \mathbb{W})$-pattern functor *is the functor $\mathbb{P}_{(\mathbb{D},\mathbb{W})} : \mathbb{D}$-**CGraph** $\to \mathbb{W}$-**Graph** defined as follows:*

- *For graph G in $\mathbb{D}$-**CGraph**, the graph $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(G)$ has the same nodes as G and, for $w \in \mathbb{W}$, a w-arc of source v_1 and target v_2 whenever there is a w-path $v_1 \overset{w}{\rightsquigarrow} v_2$ in G.*
- *For a morphism $m : G \to F$ in $\mathbb{D}$-**CGraph**, the morphism $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(m) : \mathbb{P}_{(\mathbb{D},\mathbb{W})}(G) \to \mathbb{P}_{(\mathbb{D},\mathbb{W})}(F)$ has the same node function as m and an edge function that sends the unique w-arc of source v in $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(G)$ to the unique w-arc of source v in $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(F)$.*

Proof. Let $G = (V_G, E_G, s_G, t_G, l_G) \in \mathbb{D}$-**CGraph**, then for all v in V_G and all $w \in \mathbb{W}$, there exists a unique path p of source v and labeled w in G. Therefore, the functor is well-defined on objects. Furthermore, the uniqueness of w-path in G yields uniqueness of w-arcs in $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(G)$, for each node of G.Thus, the functor is well-defined on morphisms. □

Let us consider a combinatorial graph $\mathscr{G} \in \mathbb{D}$-**CGraph**. To be able to apply a rule scheme on a graph $\mathscr{G}$, we still need to ensure that the instantiation of the left-hand side of the rule scheme

produces a graph that can be matched to $\mathcal{G}$. Therefore, we will now make precise the two following points:

- A selection mechanism should specify what part of the $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathcal{G})$ will be used as the pattern graph.
- A method should specify how to construct the match from the instantiated rule.

For the present discussion, uniqueness is always considered up to isomorphism. Let us first deal with the selection mechanism and assume we want to apply the transformation to a topological cell in an object modeled by a graph $\mathcal{G}$, i.e., a maximal subgraph built using a subset of the labeling alphabet. We may not know all the nodes in the graph affected by the transformation, but we know for sure that the nodes corresponding to the cell are concerned. If we specify one node that must be in the transformed part of the graph, then the transformed part is the subgraph of $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathcal{G})$ containing the node and all nodes reachable to and from this node. Formally, let us define P_v (for a node v in $\mathcal{G}$) as the maximal subgraph of $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathcal{G})$ such that v is in P_v and, for all arcs e such that $s(e)$ or $t(e)$ is in P_v, then e, $s(e)$, and $t(e)$ are in P_v. For all nodes v in $\mathcal{G}$, there is a unique graph P_v and a unique monomorphism $p_v\colon \mathrm{P}_v \hookrightarrow \mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathcal{G})$.

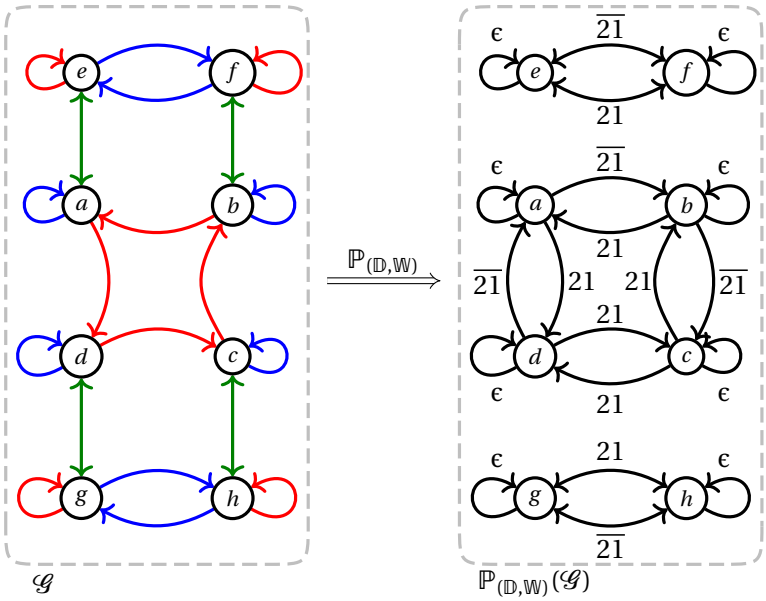


Figure 4.21: Application of the $(\{1,2\},\{\epsilon,21,\overline{21}\})$-pattern functor to a $\{1,2,3\}$-combinatorial graph.

Example 48 (Pattern functor)**.** Consider the combinatorial graph $\mathcal{G}$ depicted on the left of Figure 4.21. The application of the $(\mathbb{D},\mathbb{W})$-pattern functor for $\mathbb{W} = \{\epsilon, 21, \overline{21}\}$ yields the $\mathbb{W}$-graph on the right. The non-oriented 3-arcs are discarded. Each node is the source of an ϵ-loop and the 21 and $\overline{21}$-paths are turned into arcs. Note that the 1111-cycle involving nodes a, b, c, and d, is oriented, meaning that the 21 and $\overline{21}$-arcs on these nodes are oriented. Conversely, the 1 and 2-arcs on nodes e, f, g, and h are non-oriented and yield non oriented 21 and $\overline{21}$-arcs in $\mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathcal{G})$. The application of the pattern functor results in three strongly connected components. The top one corresponds to P_e and P_f, the middle one to P_a, P_b, P_c and P_d, and the bottom one to P_g and P_h. Note that the middle one corresponds to the pattern graph P of Figure 4.16.

For each node v from $\mathscr{G}$, the pattern graph P_v is unique. Given a graph scheme L from a rule scheme $\mathscr{S}$, the product of the embedded pattern graph $\mathbb{E}_{\mathbb{D}}(\mathrm{P}_v)$ and L is also unique. Therefore, given a node v from $\mathscr{G}$, the instantiation $\iota(\mathrm{L},\mathrm{P}_v)$ is unique. However, node v from P_v may have been duplicated in the product construction. Indeed, node v yields as many nodes in the product as nodes in L. Nevertheless, if we identify a node v_{L} among the nodes of L that we call a *hook*, the product yields a unique node (v, v_{L}).

Let us assume that the maximal subgraph of $\iota(\mathrm{L},\mathrm{P}_v)$ containing the node (v, v_{L}) is actually $\iota(\mathrm{L},\mathrm{P}_v)$, i.e., $\iota(\mathrm{L},\mathrm{P}_v)$ is connected. If there is a mono $\iota(\mathrm{L},\mathrm{P}_v) \hookrightarrow \mathscr{G}$ mapping (v, v_{L}) to v, then the morphism is unique. This uniqueness comes from the incident arcs constraint verified by $\mathscr{G}$. Indeed, each arc incident to (v, v_{L}) can only be sent to the unique arc incident to v with the same label. The mapping of the arcs incident to (v, v_{L}) provides a mapping of the nodes adjacent to (v, v_{L}). There is a unique way to propagate this mapping, and we get the uniqueness of $m\colon \iota(\mathrm{L},\mathrm{P}_v) \hookrightarrow \mathscr{G}$ such that $m((v, v_{\mathrm{L}})) = v$.

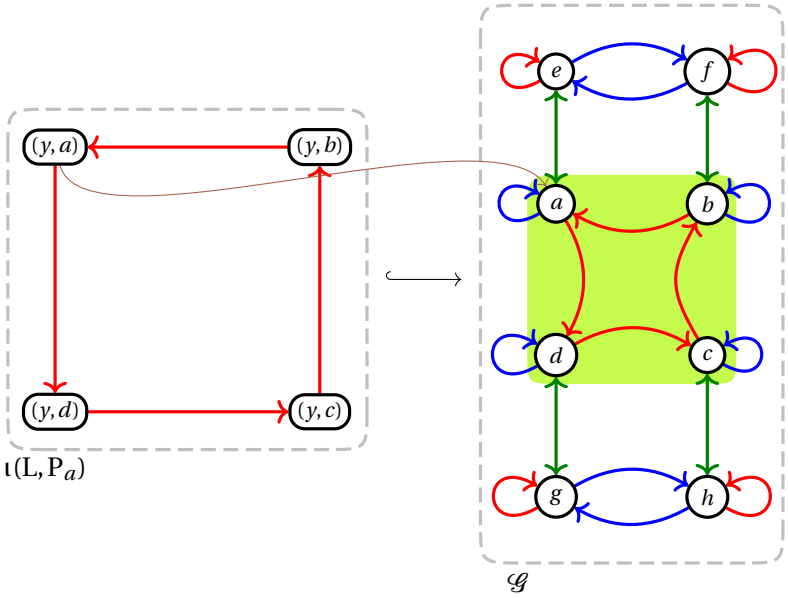


Figure 4.22: Mono from the instantiated left-hand side of the rule to the combinatorial graph.

Example 49 (Construction of the match)**.** If we identify y as a hook in the left-hand side of the rule scheme of Figure 4.20, then the node (y, a) is uniquely defined in the instantiated rule of the same figure. As we can see, there is a unique mono $\iota(\mathrm{L},\mathrm{P}_a) \hookrightarrow \mathscr{G}$ (where $\mathscr{G}$ is the combinatorial graph on the left of Figure 4.21) that maps (y, a) to a. It is the mono that maps (y, a) to a, (y, b) to b, (y, c) to c, (y, d) to d and the arcs accordingly. The assumption that the monomorphism maps (y, a) to a is represented by the thin brown arrow from (y, a) to a, and the morphism is highlighted in green in Figure 4.22.

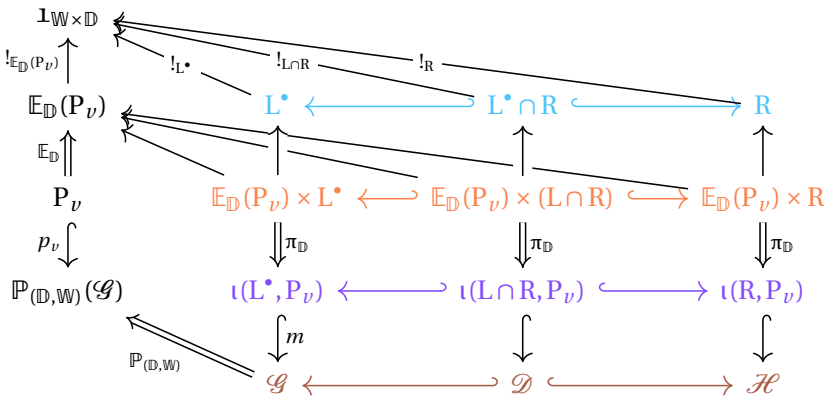
Note that if $\iota(\mathrm{L},\mathrm{P}_v)$ is not connected, several non isomorphic monos $\iota(\mathrm{L},\mathrm{P}_v) \hookrightarrow \mathscr{G}$ might exist. For instance, if two distinct components of $\iota(\mathrm{L},\mathrm{P}_v)$ are isomorphic, we can exchange their image and get another mono. This limit can be bypassed whenever the partition arises from L. In this case, if we identify one hook pair (v, v_{L}) per component of L such that all the corresponding pattern graphs P_v's are isomorphic, we restore the uniqueness of the mono. Note that the construction of the pattern graph guarantees that it is connected. The embedding functor construction ensures

that every arc incident to each node of L is associated with a unique arc in $\mathbb{E}_{\mathbb{D}}(P_\nu)$. Therefore, the specification of hook pairs is sufficient to guarantee the uniqueness of the match, should it exist. In this case, the local propagation exploiting the incident arcs property provides an algorithm to build the match.

As the instantiation mechanism is rather general, it can produce rules that may not be applicable in some cases. We could impose conditions on the labels used in the ruling scheme, but we may unnecessarily restrict our framework expressiveness. Therefore, we believe it is up to the user to check on examples that any written rule corresponds to the desired operation.

We have discussed how we can apply a rule scheme to a combinatorial graph; let us now wrap up the whole process. To apply a $(\mathbb{W}\times\mathbb{D})$-rule scheme $\mathscr{S}$ to a $\mathbb{D}$-combinatorial graph $\mathscr{G}$, we first extract a pattern graph P from $\mathscr{G}$. This pattern graph P is built in two steps: the application of the pattern functor $\mathbb{P}_{(\mathbb{D},\mathbb{W})}$ and the choice of a monomorphism $p_\nu : P_\nu \hookrightarrow \mathbb{P}_{(\mathbb{D},\mathbb{W})}(\mathscr{G})$. The choice of the monomorphism boils down to the choice of a node ν in $V_{\mathscr{G}}$ for each hook in L. This choice specifies where the operation should be realized. With this pattern graph, we instantiate the $(\mathbb{W}\times\mathbb{D})$-rule scheme to get a $\mathbb{D}$-rule via $\iota(-,P_\nu)$. Thanks to the hook nodes in L, we construct the unique mono $m\colon \iota(L,P_\nu)\hookrightarrow\mathscr{G}$ and finally realize the direct derivation $\mathscr{G}\Rightarrow^{\iota(\mathscr{S},P),m}\mathscr{H}$. An example of the complete pipeline is given in Figure 4.23, where both the hook node in the rule scheme and the chosen node in the combinatorial graph are filled in blue. The green coloring puts forward the transformation in the graph.

The complete construction can be summarized by the following commutative diagram where $L^\bullet$ means that L has a hook per connected component. The top span $L^\bullet \hookleftarrow L\cap R \hookrightarrow R$ (in blue) is the rule scheme. The second span $\mathbb{E}_{\mathbb{D}}(P_\nu)\times L^\bullet \hookleftarrow \mathbb{E}_{\mathbb{D}}(P_\nu)\times(L\cap R)\hookrightarrow \mathbb{E}_{\mathbb{D}}(P_\nu)\times R$ (in orange) is obtained by the product with $\mathbb{E}_{\mathbb{D}}(P_\nu)$, where P_ν is a subgraph of the pattern functor applied to $\mathscr{G}$. The third span (in purple) is obtained with the projecting functor and yields the last span (in brown) by standard DPO-rewriting.



As a final remark, let us point out that the set of words $\mathbb{W}$ has been given a priori for the rule scheme and the pattern graph. Up to now, $\mathbb{W}$ was just a superset of all words used to label a given $\mathbb{W}$-pattern graph or the $(\mathbb{W}\times\mathbb{D})$-graph schemes of a rule scheme. In the following three sections, we will impose conditions on $\mathbb{W}$ to lift the condition for preserving our three elementary constraints. Nonetheless, the choice of $\mathbb{W}$ inherently comes from the transformation that we want to describe through the rule scheme.

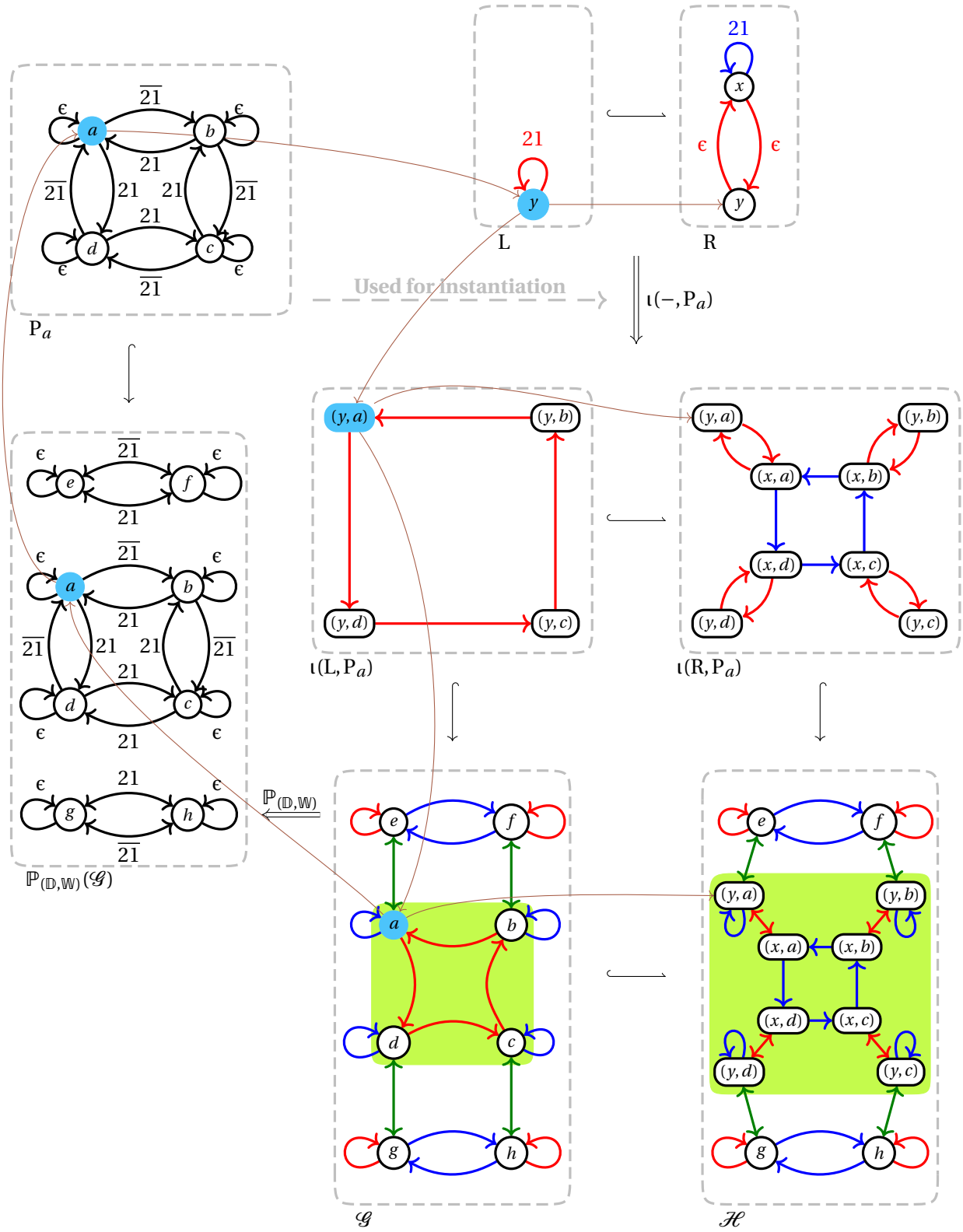


Figure 4.23: Complete process to transform a combinatorial graph with a rule scheme.

4.4 Incident arcs consistency in rule schemes

We now lift the incident arcs condition from $\mathbb{D}$-combinatorial rules to rule schemes. Since rule instantiations are essentially defined through a product, it is natural that each of the two members of the product should be subject to conditions to ensure that the resulting instantiation fulfills the intended incident arcs condition. Therefore, we will start by defining a first condition for the pattern graphs and a second for the rule schemes. Afterward, we will show that, under such hypotheses, the instantiation of a consistent rule scheme via a consistent pattern graph yields combinatorial rules.

The constraint on the pattern graph is the incident constraint for the set of words $\mathbb{W}$ considered as arc labels. Recall that a $\mathbb{W}$-pattern graph P satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$ if every node v of P is the source of a unique w-arc and the target of a unique w-arc for each word in $\mathbb{W}$. Note that when the pattern graph P is defined as a maximal subgraph of a $\mathbb{D}$-combinatorial graph G generated from a given node and the words of $\mathbb{W}$, we trivially have $\mathrm{P} \models \mathtt{I}_{\mathrm{P}}(\mathbb{W})$. For instance, the pattern graph of Figure 4.16 (from Section 4.3) satisfies the incident arcs constraint on the set of words $\{\epsilon, 21, \overline{21}\}$. Let us point out that the incident arcs constraint on pattern graphs guarantees that when building a product using such a pattern graph, all arcs of a graph scheme have a corresponding arc for each node in the embedded pattern graph. Let us now introduce a counterpart condition for rule schemes.

Definition 49 (Incident arcs condition for rule schemes)**.** *A $(\mathbb{W}\times\mathbb{D})$-rule scheme $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ satisfies the* incident arcs condition $\mathtt{I}_{\mathscr{S}}(i)$ *for a dimension i in $\mathbb{D}$ if its core $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$. In this case, we write $\mathscr{S} \models \mathtt{I}_{\mathscr{S}}(i)$. If the core $\pi_{\mathbb{D}}(\mathscr{S})$ is a $\mathbb{D}$-combinatorial rule, $\mathscr{S}$ is said to be a $(\mathbb{W} \times \mathbb{D})$-combinatorial rule scheme.*

In other words, a $(\mathbb{W} \times \mathbb{D})$-rule scheme is a $(\mathbb{W} \times \mathbb{D})$-combinatorial rule scheme if for all i in $\mathbb{D}$:

1. Any preserved node of $\pi_{\mathbb{D}}(\mathrm{L} \cap \mathrm{R})$ is the source (resp. target) of an i-arc in $\pi_{\mathbb{D}}(\mathrm{L})$ if and only if it is the source (resp. target) of an i-arc in $\pi_{\mathbb{D}}(\mathrm{R})$.
2. $\pi_{\mathbb{D}}(\mathrm{L}) \models \mathtt{I}_{(\mathrm{V_L}\setminus\mathrm{V_R}),\pi_{\mathbb{D}}(\mathrm{L})}(i)$.
3. $\pi_{\mathbb{D}}(\mathrm{R}) \models \mathtt{I}_{(\mathrm{V_R}\setminus\mathrm{V_L}),\pi_{\mathbb{D}}(\mathrm{R})}(i)$.
4. For all nodes v of $\pi_{\mathbb{D}}(\mathrm{L})$, there exists at most one arc of source, resp. target, v in $\pi_{\mathbb{D}}(\mathrm{L})$.

Let us recall that the first two sub-conditions concern the weak-incident arcs conditions. The third one is equivalent to the also called gluing condition. The last one is a necessary condition for the existence of a match and is specified to ease talking about the rules.

Example 50 (Incident arcs condition for rule schemes)**.** The rule scheme from the previous section (see Figure 4.19a) satisfies $\mathtt{I}_{\mathscr{S}}(\{1,2\})$. The core of the rule scheme is given in Figure 4.19b. Node y belongs to the interface. It is the source of a 1-loop in L. It is also the source and target of an 1-arc in R. Thus, the first sub-condition holds. There is no deleted node. The added node x is the source of a 2-loop and the source and target of a 1-arc. Thus, the second and the third sud-conditions also hold. In the end, the rule scheme is a $\{\epsilon, 21, \overline{21}\}, \{1,2\}$-combinatorial rule scheme.

Theorem 4 (Lifting the incident arcs condition to rule schemes)**.** *Let i be a dimension in $\mathbb{D}$ and $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W} \times \mathbb{D})$-rule scheme.*

The rule scheme $\mathscr{S}$ satisfies the incident arcs condition $\mathtt{I}_{\mathscr{S}}(i)$ (Definition 49) if and only if for all $\mathbb{W}$-pattern graph P such that $\mathrm{P} \models \mathtt{I}_{\mathrm{P}}(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S}, \mathrm{P})$ satisfies the incident arcs condition $\mathtt{I}_{\iota(\mathscr{S},\mathrm{P})}(i)$.

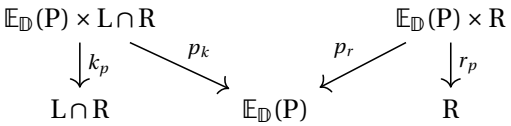
$$\mathscr{S} \models \mathtt{I}_{\mathscr{S}}(i) \iff (\forall \mathrm{P} \in \mathbb{W}\text{-}\mathbf{Graph}, \mathrm{P} \models \mathtt{I}_{\mathrm{P}}(\mathbb{W}) \implies \iota(\mathscr{S}, \mathrm{P}) \models \mathtt{I}_{\iota(\mathscr{S},\mathrm{P})}(i))$$

As an intuition, the proof holds because $\iota(-, \mathrm{P})$ is a functor that transfers the incident arcs from $\mathscr{S}$ to $\iota(\mathscr{S}, \mathrm{P})$. Indeed, the product construction guarantees that the uniqueness of an incident arc both in P and G yields a unique arc in $\iota(\mathrm{G}, \mathrm{P})$. In the reverse sense, we remark that choosing the terminal graph $\mathtt{1}_{\mathbb{W}}$ yields the expected properties.

Proof. [139] Let $\mathscr{S} = \mathrm{L} \xhookleftarrow{i_{\mathrm{L}}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_{\mathrm{R}}} \mathrm{R}$ be a $(\mathbb{W} \times \mathbb{D})$-rule scheme and $i \in \mathbb{D}$ be a dimension.

($\Longrightarrow$) Assume that the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the incident arcs condition $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$. Consider a $\mathbb{W}$-pattern graph P that satisfies the incident arcs constraint $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$.

Let p_k be the projection $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L} \cap \mathrm{R} \to \mathbb{E}_{\mathbb{D}}(\mathrm{P})$, p_r be the projection $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R} \to \mathbb{E}_{\mathbb{D}}(\mathrm{P})$, k_p be the projection $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L} \cap \mathrm{R} \to \mathrm{L} \cap \mathrm{R}$, and r_p be the projection $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R} \to \mathrm{R}$, defined from the following products:



We prove the two sub-conditions of the weak incident arcs condition (Definition 33) and the condition on the left-hand side equivalent to the gluing condition (Fact 1) to obtain the complete incident arcs condition (Definition 34).

▶ *Sub-condition 1 of the weak incident arcs condition for the rule $\iota(\mathscr{S}, \mathrm{P})$ and a dimension i.*

Let v be a preserved node of $\iota(\mathrm{L} \cap \mathrm{R}, \mathrm{P})$. Thus, v is a preserved node of $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L} \cap \mathrm{R}$ and there are two nodes a in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and u in $\mathrm{L} \cap \mathrm{R}$ such that $p_k(v) = a$ and $k_p(v) = u$. If v is the source of an i-arc in $\iota(\mathrm{L}, \mathrm{P})$ then there is a word w in $\mathbb{W}$ such that v is the source of an arc labeled (w, i) in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L}$. By construction of the product, there is an arc of source a in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and an arc of source u in L both labeled (w, i). Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, there is a word w' in $\mathbb{W}$ such that u is the source of an arc labeled (w', i) in R. Since $\mathrm{P} \models \mathtt{I}_{\mathrm{P}}(\mathbb{W})$, the embedding functor $\mathbb{E}_{\mathbb{D}}$ ensures the existence of an arc labeled (w', i) in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ of source a. Therefore, v is the source of an arc labeled (w', i) in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R}$, transformed into an i-arc in $\iota(\mathrm{R}, \mathrm{P})$ by the projecting functor.

The proof holds for the target of an i-arc in $\iota(\mathrm{L}, \mathrm{P})$ and for the source or target of an i-arc in $\iota(\mathrm{R}, \mathrm{P})$. Thereafter, any preserved node of $\iota(\mathrm{L} \cap \mathrm{R}, \mathrm{P})$ is the source (resp. target) of an i-arc in $\iota(\mathrm{L}, \mathrm{P})$ if and only if it is the source (resp. target) of an i-arc in $\iota(\mathrm{R}, \mathrm{P})$.

▶ *Sub-condition 2 of the weak incident arcs condition for the rule $\iota(\mathscr{S}, \mathrm{P})$ and a dimension i.*

Let v be an added node[2] of $\iota(\mathrm{R} \setminus \mathrm{L}, \mathrm{P})$. Thus, v is a node of $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times (\mathrm{R} \setminus \mathrm{L})$ and there are two nodes a in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and u in $\mathrm{R} \setminus \mathrm{L}$ such that $p_r(v) = a$ and $r_p(v) = u$. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs

condition $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, u is the source of a unique i-arc in $\pi_{\mathbb{D}}(\mathrm{R})$. This i-arc yields an arc labeled (w,i), for a word in w in $\mathbb{W}$, that is the only arc in R of source u having i as the second part of its label. Since P satisfies the incident arcs constraint $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$, the embedding functor $\mathbb{E}_{\mathbb{D}}$ ensures the existence and uniqueness of a (w,i)-arc of source a. Therefore, there is an arc labeled (w,i) of source v in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R}$, and it is the only arc of source v having i as the second part of its label. This arc is transformed into an i-arc in $\iota(\mathrm{R},\mathrm{P})$ by the projecting functor. Thus, there is a unique i-arc of source v in $\iota(\mathrm{R},\mathrm{P})$.

The proof holds for an i-arc of target v. Thereafter, any added node of $\iota(\mathrm{R}\setminus\mathrm{L},\mathrm{P})$ is the source (resp. target) of a unique i-arc and $\iota(\mathscr{S},\mathrm{P})$ satisfies sub-condition 2 of the weak incident arcs condition.

▶ *Gluing condition.*
The proof for a node deleted from $\iota(\mathrm{L}\setminus\mathrm{R},\mathrm{P})$ is the same as the proof for a node added in $\iota(\mathrm{R}\setminus\mathrm{L},\mathrm{P})$.

Thereafter, $\iota(\mathscr{S},\mathrm{P})$ satisfies the incident arcs condition $\mathtt{I}_{\iota(\mathscr{S},\mathrm{P})}(\mathbb{D})$.

($\Longleftarrow$) Assume that for all $\mathbb{W}$-pattern graph P that satisfies the incident arcs constraint $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S},\mathrm{P})$ satisfies the incident arcs condition $\mathtt{I}_{\iota(\mathscr{S},\mathrm{P})}(\mathbb{D})$. In particular, consider the terminal graph $\mathtt{1}_{\mathbb{W}}$. Then, $\iota(\mathscr{S},\mathtt{1}_{\mathbb{W}})$ is the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$. Thus, $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$.

Thereafter, the rule scheme $\mathscr{S}$ satisfies incident arcs condition $\mathtt{I}_{\mathscr{S}}(i)$. □

Provided that pattern graphs and rule schemes satisfy certain conditions, which can be verified by direct static analysis, then Theorem 4 states that all rules obtained by instantiation verify the incident arcs conditions.

Example 51 (Lifting the incident arcs condition to rule schemes)**.** The rule scheme from Figure 4.19 is a $(\{\epsilon, 21, \overline{21}\} \times \{1,2\})$-combinatorial rule scheme. Its instantiation with the pattern graph of Figure 4.16 yields the rule of Figure 4.20, which satisfies the incident arcs condition for the dimensions 1 and 2.

Likewise, we will study constraints on pattern graphs and conditions on rule schemes for the non-orientation and cycle properties.

4.5 Non-orientation consistency in rule schemes

By analogy with the extension of the incident arcs condition, we study the condition of non-orientation. First, we give a constraint on pattern graphs and a condition on the rule schemes such that scheme instantiation produces a rule that satisfies the non-orientation condition.

Definition 50 (Non-orientation constraint for pattern graphs)**.** *A $\mathbb{W}$-pattern graph* P *satisfies the* non-orientation constraint $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$ *if for every w in $\mathbb{W}$, $\overline{w}$ is also in $\mathbb{W}$ and every w-arc in* P *admits a reverse $\overline{w}$-arc.*

[2] Since $\mathrm{R}\setminus\mathrm{L}$ is not a graph, $\iota(\mathrm{R}\setminus\mathrm{L},\mathrm{P})$ is not application of the functor $\iota(-,\mathrm{P})$ to $\mathrm{R}\setminus\mathrm{L}$. It is used as a notation to consider added elements of $\iota(\mathscr{S},\mathrm{P})$. Similarly, we write $\iota(\mathrm{L}\setminus\mathrm{R},\mathrm{P})$ for the added elements of $\iota(\mathscr{S},\mathrm{P})$. We use the same notation for the functor $\mathbb{E}_{\mathbb{D}}$ with the product.

If the pattern graph P is built on a $\mathbb{D}$-combinatorial graph G that satisfies the non-orientation constraint $\mathtt{O}_{\mathrm{G}}(\mathbb{D})$, then P trivially satisfies the non-orientation constraint for pattern graphs $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$. For instance, the pattern graph of Figure 4.16 (see Section 4.3) satisfies the non-orientation constraint for pattern graph on the set of words $\{\epsilon, 21, \overline{21}\}$. The non-orientation constraint on pattern graphs ensures that, for all words w in $\mathbb{W}$, arcs labeled w and $\overline{w}$ in the graph scheme yields arcs and reverse arcs in the product construction.

Definition 51 (Non-orientation condition for rule schemes)**.** *A $(\mathbb{W}\times\mathbb{D})$-rule scheme $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ satisfies the* non-orientation condition $\mathtt{O}_{\mathscr{S}}(i)$ *for a dimension i in $\mathbb{D}$ if:*

- *The core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the non-orientation condition $\mathtt{O}_{\pi_{\mathbb{D}}\mathscr{S}}(i)$.*
- *For any arcs e and e' in $\mathscr{S}$ such that[3] $l_{\pi_{\mathbb{D}}(\mathscr{S})}(\pi_{\mathbb{D}}(e)) = l_{\pi_{\mathbb{D}}(\mathscr{S})}(\pi_{\mathbb{D}}(e')) = i$ and e is the reverse arc of e' in $\pi_{\mathbb{D}}(\mathscr{S})$, then $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e))$ is the conjugate of $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e'))$.*

Example 52 (Non-orientation condition for rule schemes)**.** The rule scheme given in Section 4.3 in Figure 4.19a satisfies the non-orientation condition $\mathtt{O}_{\mathscr{S}}(1)$. Indeed, its core rule (see Figure 4.19b) contains a 1-loop in L and two reverse i-arcs in R. The 1-arcs in the right-hand side of the core rule are $(1,\epsilon)$-arcs in the rule scheme, thus the parts of the label on $\mathbb{W}$ are conjugate words.

Theorem 5 (Lifting the non-orientation condition to rule schemes)**.** *Let i be a dimension in $\mathbb{D}$ and $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme.*

The rule scheme $\mathscr{S}$ satisfies the non-orientation condition $\mathtt{O}_{\mathscr{S}}(i)$ if and only if for all $\mathbb{W}$-pattern graph P *that satisfies both the incident arcs constraint $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$ and the non-orientation constraint $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S},\mathrm{P})$ satisfies the non-orientation condition $\mathtt{O}_{\iota(\mathscr{S},\mathrm{P})}(i)$.*

$$\mathscr{S} \models \mathtt{O}_{\mathscr{S}}(i) \iff (\forall \mathrm{P} \in \mathbb{W}\text{-}\mathbf{Graph}, \mathrm{P} \models \mathtt{I}_{\mathrm{P}}(\mathbb{W}) \wedge \mathrm{P} \models \mathtt{O}_{\mathrm{P}}(\mathbb{W}) \implies \iota(\mathscr{S},\mathrm{P}) \models \mathtt{O}_{\iota(\mathscr{S},\mathrm{P})}(i))$$

The proof is simpler than the one of Theorem 4 since fewer properties need to be checked. Here again, it boils down to the good behavior of the product. The reverse proof relies again on the exhibition of an appropriate graph.

Proof. [139] Let i be a dimension $\mathbb{D}$ and $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme.

($\Longrightarrow$) Assume that the rule scheme $\mathscr{S}$ satisfies the two sub-conditions of Definition 51 and consider a $\mathbb{W}$-pattern graph P that satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$ and $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$.

Without loss of generality, let us consider the case of $\iota(\mathrm{L}\setminus\mathrm{R},\mathrm{P})$. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies $\mathtt{O}_{\pi_{\mathbb{D}}\mathscr{S}}(i)$ and $\mathtt{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, $\mathrm{L}\setminus\mathrm{R}$ always contains an arc and its reverse. These arcs have conjugate words on the first part of their label. Because P satisfies $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$, the product pairs these arcs with reverse arcs of $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P}))$. Thus, the product creates reverse arcs in $\iota(\mathrm{L},\mathrm{P})$.

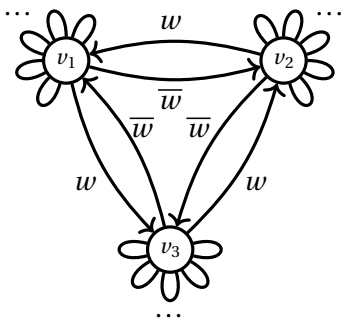
Thereafter, $\iota(\mathscr{S},\mathrm{P})$ satisfies $\mathtt{O}_{\iota(\mathscr{S},\mathrm{P})}(i)$.

[3] Where $l_{\pi_{\mathbb{D}}(\mathscr{S})}$ stands for $l_{\pi_{\mathbb{D}}(\mathrm{L})}$, $l_{\pi_{\mathbb{D}}(\mathrm{L}\cap\mathrm{R})}$ or $l_{\pi_{\mathbb{D}}(\mathrm{R})}$ (resp. $l_{\pi_{\mathbb{W}}(\mathscr{S})}$ stands for $l_{\pi_{\mathbb{W}}(\mathrm{L})}$, $l_{\pi_{\mathbb{W}}(\mathrm{L}\cap\mathrm{R})}$ or $l_{\pi_{\mathbb{W}}(\mathrm{R})}$) depending on whether $e\in\mathrm{L}$, $e\in\mathrm{L}\cap\mathrm{R}$ or $e\in\mathrm{R}$.

($\Longleftarrow$) Assume that for all $\mathbb{W}$-pattern graph P that satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$ and $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S},\mathrm{P})$ satisfies the non-orientation condition $\mathtt{O}_{\iota(\mathscr{S},\mathrm{P})}(i)$. Similar to the proof of theorem 4, the terminal graph $\mathtt{1}_{\mathbb{W}}$ gives us the condition on the core of $\mathscr{S}$.

For the second condition, consider an i-arc e in $\mathscr{S}$ such that $\pi_{\mathbb{D}}(e)$ is an arc constrained by $\mathtt{O}_{\pi_{\mathbb{D}}\mathscr{S}}(i)$. Note that since $\mathscr{S}$ is a $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme, the reverse arc of $\pi_{\mathbb{D}}(e)$ is unique (theorem 4). Let us assume that $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e)) = w$. Again, we build a scheme instantiation for $\mathscr{S}$ using a particular graph.

Intuitively, we need a graph pattern G_w with a w-arc from a node v_1 to a node v_2 such that there a unique arc from v_2 to v_1. The non-orientation constraint of pattern graphs imposes that this arc is labeled $\overline{w}$. Then, the instantiation $\iota(\mathscr{S},\mathrm{G}_w)$ will yields i-arcs images of both the w-arc and e. Using the non-orientation condition on the instantiated rule, we get the non-orientation condition on the rule scheme. Since G_w has to satisfy both the incident arcs constraint and non-orientation constraint of pattern graphs, it needs at least three nodes to fulfill the previous requirement. With one node, there would be loops for each word in $\mathbb{W}$. With two nodes, the incident arcs constraint imposes an w-arc of source v_2, which can only have v_1 as target. In this case there would be two arcs from v_2 to v_1, one labeled $\overline{w}$ and one labeled w. There is a solution with three nodes. Intuitively, G_w is composed of 3 occurrences of $\mathtt{1}_{\mathbb{W}\setminus\{w,\overline{w}\}}$ glued together with a www-cycle, like this :



Formally[4], let $\mathrm{G}_w = (\mathrm{V}_w, \mathrm{E}_w, s_w, t_w, l_w)$ be the graph such that :

- $\mathrm{V}_w = \{v_1, v_2, v_3\}$.
- $\mathrm{E}_w = \{e_{1,w}, e_{1,\overline{w}}, e_{2,w}, e_{2,\overline{w}}, e_{3,w}, e_{3,\overline{w}}\} \cup \mathrm{E}_1 \cup \mathrm{E}_2 \cup \mathrm{E}_3$ where $\mathrm{E}_k = \{e_{k,w'} \mid w' \in \mathbb{W}\setminus\{w,\overline{w}\}\}$ for $k = 1$, 2 or 3.
- $s_w(e_{k,w}) = v_k$, $s_w(e_{k,\overline{w}}) = v_{k+1}$, and $s_w(e \in \mathrm{E}_k) = v_k$ for $k = 1$, 2 or 3.
- $t_w(e_{k,w}) = v_{k+1}$, $t_w(e_{k,\overline{w}}) = v_k$, and $t_w(e \in \mathrm{E}_k) = v_k$ for $k = 1$, 2 or 3.
- For any $1 \le k \le 3$, for any $w' \in \mathbb{W}$, $l_w(e_{k,w'}) = w'$.

G_w satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$ and $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$. In $\iota(\mathscr{S},\mathrm{G}_w)$, e yields 3 arcs e_1, e_2, and e_3 labeled (w,i) and such that, for $k = 1$, 2 or 3: $s_{\iota(\mathscr{S},\mathrm{G}_w)}(e_k) = (v_k, s_{\mathscr{S}}(e))$ and $t_{\iota(\mathscr{S},\mathrm{G}_w)}(e_k) = (v_{k+1}, t_{\mathscr{S}}(e))$.

Because the scheme instantiation $\iota(\mathscr{S},\mathrm{G}_w)$ satisfies the non-orientation condition $\mathtt{O}_{\iota(\mathscr{S},\mathrm{G}_w)}(i)$, the arcs e_1, e_2, and e_3 admit reverse arcs e_1^{-1}, e_2^{-1}, and e_3^{-1}. By construction of the product, they correspond to a $(\overline{w},i)$-arc in $\mathscr{S}$. Call it a. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies $\mathtt{O}_{\pi_{\mathbb{D}}\mathscr{S}}(i)$, $\pi_{\mathbb{D}}(a)$ is a reverse arc of $\pi_{\mathbb{D}}(e)$.

[4] In the definition of G_w, all the arithmetic operations are described modulo 3

By unicity, $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e))$ is the conjugate of $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(a))$ and $a = e^{-1}$.

Thereafter, the rule scheme $\mathscr{S}$ satisfies the non-orientation condition $\mathtt{O}_{\mathscr{S}}(i)$. □

Example 53 (Lifting the non-orientation condition to rule schemes)**.** The rule scheme of Figure 4.19 satisfies the $\mathtt{O}_{\mathscr{S}}(1)$ while the pattern graph of Figure 4.16 satisfies $\mathtt{O}_{\mathrm{P}}(\{\epsilon, 21, \overline{21}\})$. Thus, the instantiated rule of Figure 4.20 satifies the non-orientation condition for $\mathbb{D}$-rules $\mathtt{O}_r(1)$.

We are left with the study of the cycle property for rule schemes.

4.6 Cycles consistency in rule schemes

Lifting the incident arcs and non-orientation conditions from rules to rule schemes was pretty straightforward. However, extrapolating the cycle condition relies on verifying whether a cycle in the scheme will be instantiated into a cycle. The three sub-conditions of the cycle condition for an instantiated rule relate to the arcs in $\iota(\mathscr{S}, \mathrm{P})$. The issue is thus to be able to verify if a cycle in L or R yields a cycle in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L}$ or $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R}$ without any prior knowledge on P. Intuitively, a pattern graph and a graph scheme can be seen as two orthogonal spaces: a path containing only moves in the graph scheme stays on the same node in the pattern graph and reciprocally. Thus, we understand that a path in the product of two graphs is a cycle if and only if it is a cycle in both graphs.

4.6.1 Global constraints on combinatorial graphs

The possibility of oriented arcs complicates the identification of cycles at the level of the rule scheme. To bypass this difficulty, we broaden the local study to a global one and fully state the conditions on all the dimensions.

First, we split the dimension set $\mathbb{D}$ between the oriented dimensions $\mathbf{O}$ and the non-oriented dimension $\mathbf{N}$ so that[5] $\mathbf{O} \sqcup \mathbf{N} = \mathbb{D}$ (meaning $\overline{\mathbb{D}} = \mathbf{N} \sqcup \mathbf{O} \sqcup \overline{\mathbf{O}}$). Therefore, conjugation becomes an involution on $\overline{\mathbb{D}}$ such that $\overline{\overline{d}} = d$ for d in $\mathbf{O} \sqcup \overline{\mathbf{O}}$ and $\overline{d} = d$ for d in $\mathbf{N}$. Furthermore, we denote $\|d\|$ the integer value of d, ie $\|d\| = d$ if d is in $\mathbb{D}$ and $\|d\| = \overline{d}$ if d is in $\overline{\mathbf{O}}$.

In the sequel, we will consider $\mathbb{D} \subseteq \mathbb{N}$, split into $\mathbb{D} = \mathbf{O} \sqcup \mathbf{N}$, and $\mathbf{E} \subseteq \{(i, j) \in \mathbb{D}^2 \mid i < j\}$ as the set of exchangeable dimensions, i.e., dimensions of concern for the cycle property defined in Section 3.2.1.

Example 54 (Set of dimensions, non-oriented dimensions, exchangeable dimensions)**.** For illustrations and discussions in this section, we will use the set of integers between 0 and 3 (included), written as 0..3, for the set of dimensions. We will further assume that $\mathbf{N} = \{1, 3\}$, while $\mathbf{E} = \{(0,1), (0,2), (1,2), (1,3)\}$. Therefore, $\overline{1} = 1$, $\overline{\overline{0}} = 0$, $\|\overline{2}\| = \|2\| = 2$, and $\mathbf{O} = \{0, 2\}$.

4.6.2 Path equivalence in combinatorial graphs

In this section, we will introduce a rewriting system on $\overline{\mathbb{D}}^*$. For a complete study of string rewriting systems, we advise reading the first two chapters of [23]. Note that DPO rules define rewriting systems on graphs (or, more generally, on adhesive categories). Let us recall that a set A and a binary relation $\rightarrow$ on A define an abstract reduction system $(\mathrm{A}, \rightarrow)$. $\xrightarrow{+}$ is the transitive closure of $\rightarrow$ and

[5] Where $\mathrm{A} \sqcup \mathrm{B}$ is the disjoint union of the sets A and B.

$\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$. An abstract reduction system is confluent whenever two elements obtained from the same ancestor have a common descendant. The system is noetherian if it contains no infinite reduction chain. A string rewriting system on an alphabet Σ is a binary relation on Σ^* and defines a reduction system via sub-string rewriting.

Definition 52 (Conjugate rewriting system)**.** *The* conjugate rewriting system *for* $\mathbf{N}$ *and* $\mathbf{E}$ *is:*

$$\mathrm{R}_{\mathbf{N},\mathbf{E}} = \{(i\,\overline{i},\epsilon) \mid i \in \overline{\mathbb{D}}\} \cup \{(ji,\overline{i}\,\overline{j}) \mid (i,j) \in (\mathbf{N}\cup\mathbf{O})^2 \cup (\mathbf{N}\cup\overline{\mathbf{O}})^2, (\|i\|,\|j\|) \in \mathbf{E}\}.$$

Example 55 (Conjugate rewriting system)**.**

We discuss the conjugate rewriting system $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ built with the sets $\mathbf{N}$ and $\mathbf{E}$ from Example 54. We use the notation $u \rightarrow v$ for a pair (u,v) in $\mathrm{R}_{\mathbf{N},\mathbf{E}}$.

The first kind of rule has the form $i\,\overline{i} \rightarrow \epsilon$ where i is either a dimension or the conjugate of a dimension. For a non-oriented dimension i, we obtain a unique rule $ii \rightarrow \epsilon$, since $\overline{i} = i$. In our case we obtain the two rules $11 \rightarrow \epsilon$ and $33 \rightarrow \epsilon$. For an oriented dimension i, since $\overline{\overline{i}} = i$, we obtain two rules $i\,\overline{i} \rightarrow \epsilon$ and $\overline{i}\,i \rightarrow \epsilon$. With the two oriented dimensions 0 and 2, we obtain the rules $0\,\overline{0} \rightarrow \epsilon$, $\overline{0}\,0 \rightarrow \epsilon$, $2\,\overline{2} \rightarrow \epsilon$, and $\overline{2}\,2 \rightarrow \epsilon$.

The second kind of rule has the form $ji \rightarrow \overline{i}\,\overline{j}$, meaning that it flips two dimensions and conjugates them. However, there are some hypotheses on i and j. The pair (i,j) should belong to the set of exchangeable dimensions, and the dimensions should be in the same state (both conjugated or both non-conjugated) in the left-hand side. In our case, we have four pairs of exchangeable dimensions. Once again, we must consider whether a dimension belongs to the set of oriented or non-oriented dimensions. For instance, the rule $10 \rightarrow \overline{0}\,\overline{1}$ can be simplified to $10 \rightarrow \overline{0}\,1$, while the rule $\overline{1}\,\overline{0} \rightarrow 01$ yields the rule $1\,\overline{0} \rightarrow 01$. Similarly, the pair $(1,2)$ in $\mathbf{E}$ provides the rules $21 \rightarrow 1\,\overline{2}$ and $\overline{2}\,1 \rightarrow 12$ since 2 is also oriented. In the case of the pair $(1,3)$, both dimensions are non-oriented, and we obtain the unique rule $31 \rightarrow 13$. When both dimensions are oriented, such as for the pair $(0,2)$, only two possibilities out of the four are considered. The rules $20 \rightarrow \overline{0}\,\overline{2}$ and $\overline{2}\,\overline{0} \rightarrow 02$ are valid since 2 and 0 are both conjugated or both non-conjugated in the left-hand side. Conversely, the rules $2\,\overline{0} \rightarrow 0\,\overline{2}$ and $\overline{2}\,0 \rightarrow \overline{0}\,2$ do not belong to $\mathrm{R}_{\mathbf{N},\mathbf{E}}$.

Starting from the word, we obtain the reduction

$$\begin{aligned}
\overline{2}3130201 &\xrightarrow{(31\rightarrow 13)} \overline{2}1330201\\
&\xrightarrow{(33\rightarrow \epsilon)} \overline{2}10201\\
&\xrightarrow{(20\rightarrow \overline{0}\,\overline{2})} \overline{2}10\,\overline{0}\,\overline{2}1\\
&\xrightarrow{(0\,\overline{0}\rightarrow \epsilon)} \overline{2}1\,\overline{2}1\\
&\xrightarrow{(\overline{2}1\rightarrow 12)} 12\,\overline{2}1\\
&\xrightarrow{(2\,\overline{2}\rightarrow \epsilon)} 11\\
&\xrightarrow{(11\rightarrow \epsilon)} \epsilon
\end{aligned}$$

The conjugate rewriting system encapsulates path reduction only based on their label. A word can be reduced to another if both words label paths with the same endpoints. The first part of the set definition of $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ states that a possible simplification is the removal of two consecutive conjugate labels. The simplification corresponds to the traversal of an arc and its reverse when the label is in $\mathbf{N}$ and to the traversal of the same arc back and forth when the label belongs to $\mathbf{O}$.

The second part of the set definition classifies which two consecutive dimensions can be inverted, exploiting the cycle constraint on the underlying graph.

Lemma 9. *For two words w and w' in $\overline{\mathbb{D}}^*$, we denote $\mathbf{P}(w,w')$ the following property: For any combinatorial graph G satisfying $\mathtt{O}_G(\mathbf{N})$ and $\mathtt{C}_G(\mathbf{E})$, if there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ then there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$.*

Let w and w' be two words of $\overline{\mathbb{D}}^*$. Then, we have :

$$w \xrightarrow{*}_{\mathrm{R_{N,E}}} w' \implies \mathbf{P}(w,w')$$

The proof results from the fact that the rewriting system adequately encodes the cycle and (non-)orientation properties in the graph.

Proof. [139] Let G be a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{O}_G(\mathbf{N})$ and $\mathtt{C}_G(\mathbf{E})$, and let v be a node of G. Let us show the result of the lemma by induction on the number of steps in the reduction.

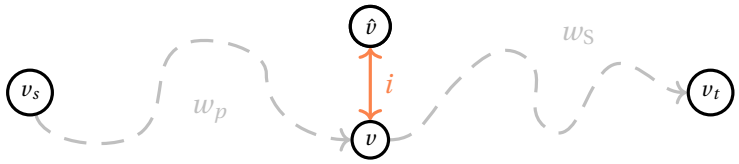
If there are no steps, $w = w'$ and the result is trivial. Otherwise, there exists a sequence of reductions:

$$w = w_0 \rightarrow_{\mathrm{R_{N,E}}} w_1 \rightarrow_{\mathrm{R_{N,E}}} w_2 \rightarrow_{\mathrm{R_{N,E}}} \dots \rightarrow_{\mathrm{R_{N,E}}} w_k = w'$$

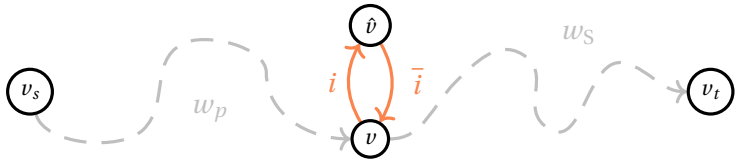
for some $k \geq 1$. Suppose there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ in G.

- If the reduction from w to w_1 is of the form $(i\,\overline{i},\epsilon)$ (with i in $\overline{\mathbb{D}}$), there exists w_p and w_s in $\overline{\mathbb{D}}^*$ such that $w = w_p i\,\overline{i}\,w_s$ and $w_1 = w_p w_s$. Let v be the target of the w_p-path starting at v_s. Since v satisfies $\mathtt{I}_v(i)$, there exists a node $\hat{v}$ and an i-arc e such that $s(e) = v$ and $t(e) = \hat{v}$. Besides G satisfies $\mathtt{O}_G(\mathbf{N})$. If i is in $\mathbf{N}$ then e admits a reverse i-arc e', otherwise $\overline{i}$ corresponds to the reverse traversal of e. Either way, $i\,\overline{i}$ is a cycle and the target of the $i\,\overline{i}$ path of source v is v. Therefore, the target of the w_p path of source v is v_s. We can remove the $i\,\overline{i}$-cycle and the target of the $w_p w_s$-path of source v_s is v_t.

  The case where i belongs to $\mathbf{N}$ and is a non-oriented dimension is illustrated in the following figure:



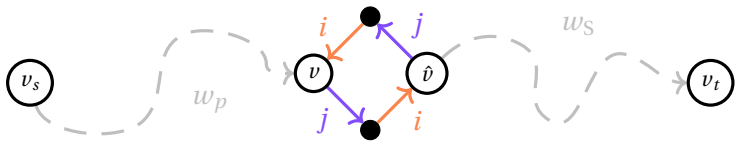
  The case where i belongs to $\mathbf{O}$ and is an oriented dimension is illustrated in the following figure:



- If the reduction from w to w_1 is of the form $(ji, \overline{ji})$ (with $(\|i\|, \|j\|)$ in $\mathbf{E}$), there exists w_p and w_s in $\overline{\mathbb{D}}^*$ such that $w = w_p ji\, w_s$ and $w_1 = w_p\, \overline{i}\,\overline{j}\, w_s$. Let v be the target of the w_p-path starting at v_s, and $\hat{v}$ be the target of the ji-path starting at v. Because the w-path starting

at v_s has v_t for target, the target of the w_s-path from $\hat{v}$ is v_t. Since G satisfies $\mathtt{C}_G(\mathbf{E})$, v is the source of a $jiji$-cycle. By unicity of the i and j arcs, the cycle contains $\hat{v}$, yielding a ji-path from $\hat{v}$ to v. The reverse traversal of this path is an $\overline{ji}$-path from v to $\hat{v}$. Therefore, the target of the $\overline{i}\,\overline{j}$-path starting at v is $\hat{v}$ and we can conclude that the target of the path labeled $w = w_p\,\overline{i}\,\overline{j}\,w_s$ starting at v_s is v_t.

The most generic case with both i and j being oriented is illustrated in the following figure:



By induction, since $w_1 \xrightarrow{*}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}} w_k = w'$, there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$ in G.

□

Let us point out that $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ is noetherian, i.e., terminates for any starting word. However, for $|\mathbf{O}| > 1$, $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ is not confluent.

Lemma 10. $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ *is noetherian.*

We use a standard approach to prove that the rewriting system is noetherian, namely building a well-founded ordering which is admissible for our rewriting system.

Proof. [139] Consider the ordering $\succ$ on $\overline{\mathbb{D}}$ such that

$$\forall d, d' \in \overline{\mathbb{D}}, d \succ d' \iff \|d\| > \|d'\| \text{ or } (\|d\| = \|d'\| \text{ and } d \in \mathbf{O}).$$

The lexicographical extension $\succ_{lex}$ of $\succ$ to $\overline{\mathbb{D}}^*$ compares words letters by letter. This extension can be further extended to the length-lexicographical ordering $\succ_{ll}$, which first compares the length of the words and then compares the words with the lexicographical ordering if the length is not sufficient to decide. In other words, the length-lexicographical ordering $\succ_{ll}$ is such that

$$\forall w, w' \in \overline{\mathbb{D}}^*, w \succ_{ll} w' \iff |w| > |w'| \text{ or } (|w| = |w'| \text{ and } w \succ_{lex} w').$$

The length-lexicographical ordering $\succ_{ll}$ is admissible for $\mathrm{R}_{\mathbf{N},\mathbf{E}}$. Indeed, for any x, y, w, w' in $\overline{\mathbb{D}}^*$, $w \succ_{ll} w'$ implies that $xwy \succ_{ll} xw'y$. Besides, the length-lexicographical extension of a well-founded ordering is a well-founded ordering [23, Chap. 2]. Finally, for all $(l, r) \in \mathrm{R}_{\mathbf{N},\mathbf{E}}$, it holds that $l \succ_{ll} r$. Therefore, $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ is noetherian. □

Lemma 11. $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ *is not confluent.*

The confluence of noetherian rewriting systems can be studied from the analysis of its critical pairs, showing local confluence, which implies confluence. Therefore, it is sufficient to exhibit a critical pair that cannot be resolved.

Proof. Since $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ is noetherian, the confluence is equivalent to the local confluence. In other words, it suffices to show that any two words directly derived from the same word have a common

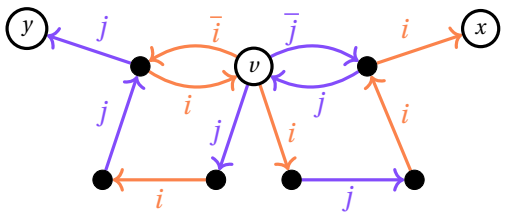
descendant to obtain the confluence of the rewriting system. Furthermore, in the case of noetherian string rewriting systems, the local confluence is decidable and can be checked by analyzing critical pairs [23, Chap. 2]. If all critical pairs are solvable, then the system is confluent; otherwise, it is not. Consider three distinct dimensions i, j, and k in $\overline{\mathbb{D}}$ such that (i,j) and (j,k) are in $\mathbf{E}$, i and k are in $\mathbf{O}$. We consider the rewriting rules $kj, \overline{j}\,\overline{k}$ and $ji, \overline{i}\,\overline{j}$ on the word kji. We obtain the words $\overline{j}\,\overline{k}\,i$ and $k\,\overline{i}\,\overline{j}$. Since i and k are in $\mathbf{O}$, both words are in normal form, i.e., they cannot be rewritten. Thus, $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ is not locally confluent and, therefore, not confluent. □

Example 56 (Non-confluence of $\mathrm{R}_{\mathbf{N},\mathbf{E}}$)**.** Reusing the rewriting system from Example 55, it holds that $210 \to 1\overline{2}0$ and $210 \to 2\overline{0}1$. Both words are in normal form since the rule $2\overline{0} \to 0\overline{2}$ and $\overline{2}0 \to \overline{0}2$ do not belong to $\mathrm{R}_{\mathbf{N},\mathbf{E}}$.

Note that we cannot extend the rewriting system to exchange any two dimensions in $\mathbf{E}$, regardless of whether they are in $(\mathbf{N} \cup \mathbf{O})$ or in $(\mathbf{N} \cup \overline{\mathbf{O}})$. In other words, the system $\mathrm{R}'_{\mathbf{N},\mathbf{E}}$ define as follows does not properly encode the cycle property.

$$\mathrm{R}'_{\mathbf{N},\mathbf{E}} = \{(i\,\overline{i}, \epsilon) \mid i \in \overline{\mathbb{D}}\} \cup \{(ji, \overline{i}\,\overline{j}) \mid (i,j) \in \overline{\mathbb{D}}^2, (\|i\|, \|j\|) \in \mathbf{E}\}$$

Indeed, $\overline{j}\,i, \overline{i}\,j$ is not a valid reduction, as shown in the following figure:



From v, we go to x by $\overline{j}\,i$ and to y by $\overline{i}\,j$, with no guarantee that Indeed, cycles can be formed with the assumption that $x \neq y$. Intuitively, we are simply stating that $\overline{j}\,i\,\overline{i}\,j$ does not form a cycle.

Note that the essential part of this lemma is about the possibility of switching two dimensions, which is a lot simpler without oriented dimensions. From this lemma, we can derive straightforward corollaries:

1. If $w \overset{*}{\longleftrightarrow}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}} w'$ then, in any combinatorial graph G satisfying $\mathtt{O}_{\mathrm{G}}(\mathbf{N})$ and $\mathtt{C}_{\mathrm{G}}(\mathbf{E})$, there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ if and only if there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$.

2. If $w \overset{*}{\longrightarrow}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}} \epsilon$, then, in any combinatorial graph G satisfying $\mathtt{O}_{\mathrm{G}}(\mathbf{N})$ and $\mathtt{C}_{\mathrm{G}}(\mathbf{E})$, a w-path is a cycle.

3. These results extends to words $\hat{w}$ and $\hat{w}'$ in $\mathbb{W}^*$, we consider the flatten word in $\overline{\mathbb{D}}^*$ to use the reduction system.

For the third corollary, a word $\hat{w}$ in $\mathbb{W}^*$ is understood as the concatenation of words from $\mathbb{W}$. Thus, for such a word $\hat{w}$, there exists $k \geq 0$ and $w_1, w_2, \dots w_k$ in $\mathbb{W}$ such that $\hat{w} = w_1 w_2 \dots w_k$. Each w_i (for $1 \leq i \leq k$) is a word of $\mathbb{W}$, i.e., a word with letters from $\overline{\mathbb{D}}$, such that $w_i = (w_i)_{(1)} \dots (w_i)_{(l_i)}$. When flatten, $\hat{w}$ is therefore equal to $(w_1)_{(1)} \dots (w_1)_{(l_1)} (w_2)_{(1)} \dots (w_2)_{(l_2)} \dots (w_k)_{(1)} \dots (w_k)_{(l_k)}$.

4.6.3 Preservation of consistency in rule schemes

The result of Lemma 9 can also be extended to $\mathbb{W}$-pattern graphs obtained via the $\mathbb{W}$-pattern functor on $\mathbb{D}$-combinatorial graphs. A word $\hat{w}$ in $\mathbb{W}^*$ labels a path $v_s \rightsquigarrow v_t$ in a $\mathbb{W}$-pattern graph P if and only if the flatten word labels a path $v_s \rightsquigarrow v_t$ in the underlying $\mathbb{D}$-combinatorial graph G. Indeed, the $\mathbb{W}$-pattern functor extracts a subset of the node set of G and turns w-paths (for w in $\mathbb{W}$) into w-arcs. Therefore, we can reduce flatten words of $\mathbb{W}^*$ using $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ and consider paths in P.

Definition 53 (Cycle constraint for pattern graphs)**.** *A $\mathbb{W}$-pattern graph* P *satisfies the* cycle constraint $\mathtt{C}_\mathrm{P}(\mathbf{N},\mathbf{E})$ *if it is coherent with* $\mathrm{R}_{\mathbf{N},\mathbf{E}}$*: for all words w, w' in $\mathbb{W}^*$, w can be rewritten as w' by* $\mathrm{R}_{\mathbf{N},\mathbf{E}}$ *implies that for any w-path $v_s \overset{w}{\rightsquigarrow} v_t$ in* P*, there is a w'-path $v_s \overset{w'}{\rightsquigarrow} v_t$ in* P*.*

The cycle constraint for pattern graphs is quite restrictive. Yet, if the pattern graph P is built on a $\mathbb{D}$-combinatorial graph that satisfies $\mathtt{O}_\mathrm{G}(\mathbf{N})$ and $\mathtt{C}_\mathrm{G}(\mathbf{E})$, then P satisfies the cycle constraint for pattern graphs $\mathtt{C}_\mathrm{P}(\mathbf{N},\mathbf{E})$ as a consequence of Lemma 9.

Example 57 (Cycle constraint for pattern graphs)**.** The pattern graph given in Figure 4.16 (see Section 4.3) satisfies the non-orientation constraint for the set of exchangeable dimensions $\{(1,2)\}$ and $\{1\}$ as the set of non-oriented dimensions.

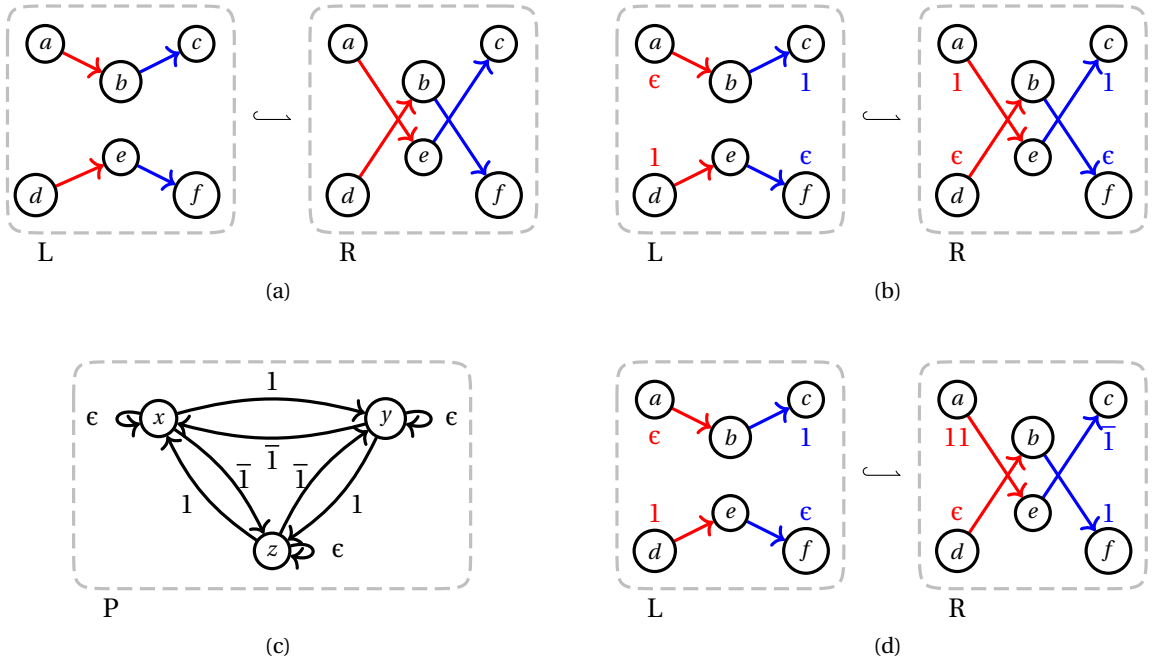


Figure 4.24: Two rule schemes with the same core and a pattern graph for instantiation.

Example 58 (Intuition for cycle preservation in rule schemes)**.** Similar to the incident arcs and the non-orientation properties, we force the rule's core to satisfy the cycle condition for $\mathbb{D}$-rules. Consider the rule given in Figure 4.24a. It satisfies $\mathtt{C}_r(1,2)$ since the two 12-paths are strongly coherent (Definition 39), and each arc belongs to an optimal path (Definition 38). Recall from Chapter 3 that an optimal path is a maximal path with the properties of being alternating, not overlapping, and only containing added or deleted arcs. The path should also not belong to a cycle. The two rules from Figure 4.24b and 4.24d have the previous $\mathbb{D}$-rule as core rule. We consider their instantiation with the pattern graph given in Figure 4.24c.

The instantiation of the rule scheme from Figure 4.24b is given in Figure 4.25a. The coherent optimal paths in the rule's core do not yield coherent paths in the instantiated rule. During the instantiation process, the product of the graph scheme with the embedded pattern graph creates an arc between nodes whenever there is an arc with the same label in both graphs. Besides, arcs in the pattern graph stand for paths in the underlying combinatorial graph. Thus, the $\mathbb{W}$-part of the label in the graphs of the rule scheme indirectly represents the path in a combinatorial graph. In the instantiated rule's left-hand side, we can find a 12-optimal path from (x,a) to (y,c), but we cannot find a coherent optimal path in the rule's right-hand side. Essentially the $\mathbb{W}$-parts of the label misfit and provide different paths. Indeed, the path abc in L (of Figure 4.24b) has 1 for $\mathbb{W}$-label whereas the path aec in R has 11 for $\mathbb{W}$-label. From Lemma 9, we know that we can prevent this from happening if we force strongly coherent paths to have their $\mathbb{W}$-part of label congruent for $\overset{*}{\leftrightarrow}_{R_{N,E}}$.

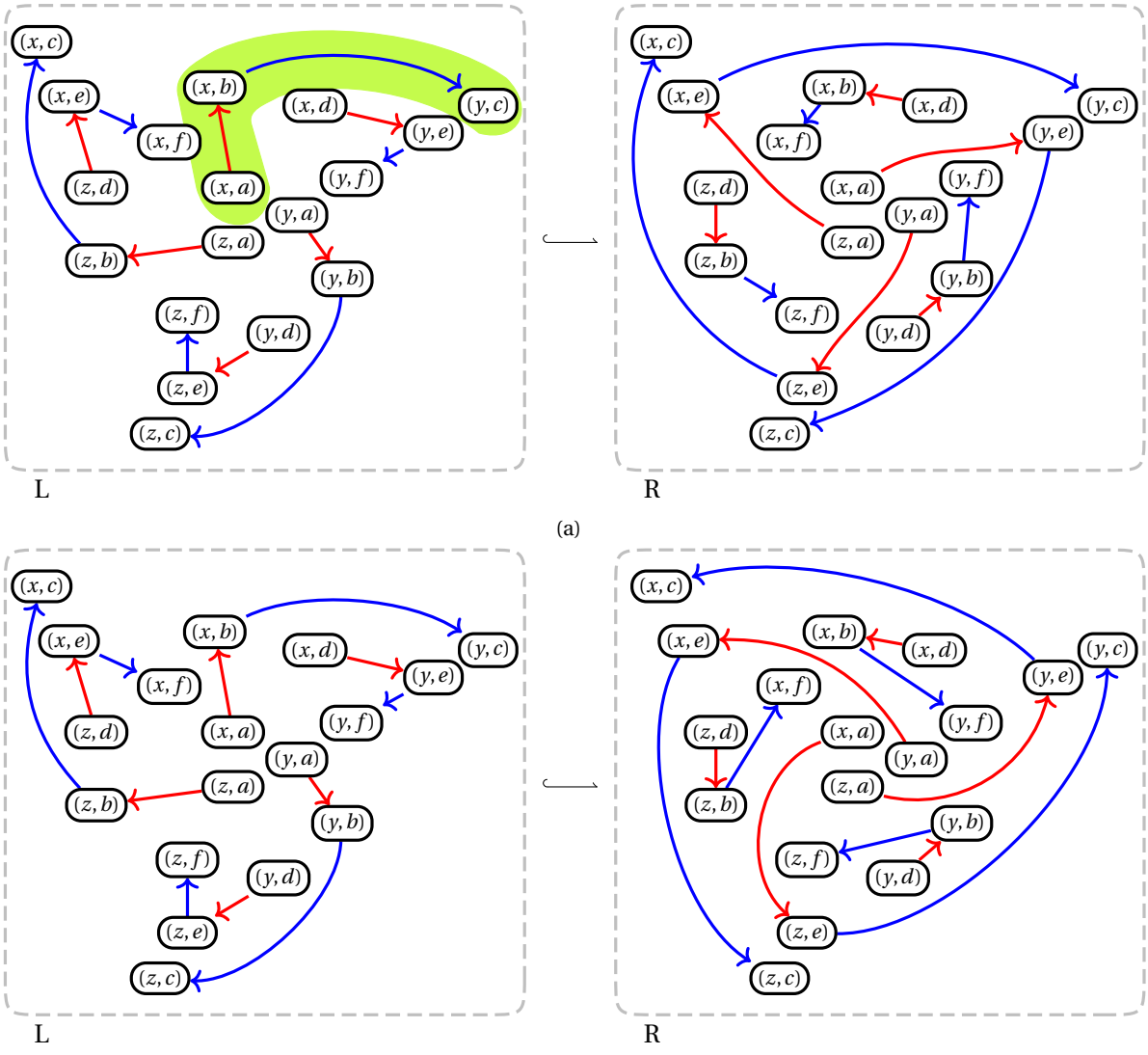


Figure 4.25: Intuition for cycle preservation in rule schemes: (a) the instantiation of the rule scheme from Figure 4.24b with the pattern graph of Figure 4.24c yields an inconsistent rule, while (b) the instantiation of the rule scheme from Figure 4.24b with the same pattern graph yields a consistent one.

We extend the definition of (strong) coherence of optimal paths to rule schemes.

Definition 54 (Coherence of optimal path in rule schemes)**.** *Let $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme. Two paths p and p' optimal in $\pi_{\mathbb{D}}(\mathscr{S})$ are* coherent *in $\mathscr{S}$ if p and p' are coherent in $\pi_{\mathbb{D}}(\mathscr{S})$ and the $\mathbb{W}$-parts of their label are congruent for $\overset{*}{\leftrightarrow}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}}$, i.e.,*

$$l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p)) \overset{*}{\leftrightarrow}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}} l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p')).$$

Similar to Definition 39, we obtain a strongly coherent *optimal path when it is coherent with a unique, coherent optimal path in the other side of the rule scheme.*

Example 59 (Coherence of optimal path in rule schemes)**.** The rule scheme of Figure 4.24d has coherent optimal paths. For instance, the 12-path abc has 1 for $\mathbb{W}$-label in L whereas the 12-path aec has $11\overline{1} \overset{*}{\rightarrow}_{\mathrm{R}_{\emptyset,\{(1,2)\}}} 1$ for $\mathbb{W}$-label in R. Similarly, the paths def (in L) and dbf (in R) have 1 for $\mathbb{W}$-label. The instantiated rule of Figure 4.25b preserves the coherence of optimal paths. For example, the optimal path $(x,a)(x,b)(y,c)$ in L is coherent with the optimal path $(x,a)(z,e)(y,c)$ in R. Likewise, the optimal path $(y,d)(z,e)(z,f)$ in L is coherent with the optimal path $(y,d)(y,b)(z,f)$ in R. These paths are strongly coherent since they have a unique, coherent counterpart.

The cycle constraint on pattern graphs ensures that cycles from graph schemes will be associated with cycles in the embedded pattern graph, yielding cycles in the instantiated rule.

Definition 55 (Cycle condition for rule schemes)**.** *A $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ satisfies the* cycle condition $\mathtt{C}_{\mathscr{S}}(\mathbf{E})$ *if:*

- *The core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathtt{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$.*
- *$\mathscr{S}$ is coherent with $\mathrm{R}_{\mathbf{N},\mathbf{E}}$, i.e.:*
  - *For any (i,j) in $\mathbf{E}$, any strongly coherent (i,j)-optimal path of $\pi_{\mathbb{D}}(\mathscr{S})$ is strongly coherent in $\mathscr{S}$.*
  - *For any (i,j) in $\mathbf{E}$, any concatenated word w labeling a cycle in $\pi_{\mathbb{W}}(\mathscr{S})$ corresponding to an $ijij$-cycle in $\pi_{\mathbb{D}}(\mathscr{S})$ can be reduced to ϵ using $\mathrm{R}_{\mathbf{N},\mathbf{E}}$.*

Example 60 (Cycle condition for rule schemes)**.** The rule scheme of Figure 4.24d satisfies the cycle condition $\mathtt{C}_{\mathscr{S}}(\{(1,2)\})$ as it contains only strongly coherent optimal paths.

Theorem 6 (Lifting the cycle condition to rule schemes)**.** *Let $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W}\times\mathbb{D})$-combinatorial rule scheme satisfying the non-orientation condition $\mathtt{O}_{\mathscr{S}}(\mathbf{N})$.*

If the rule scheme $\mathscr{S}$ satisfies the cycle condition $\mathtt{C}_{\mathscr{S}}(\mathbf{E})$, then for all $\mathbb{W}$-pattern graph P that satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$, and $\mathtt{C}_{\mathrm{P}}(\mathbf{N},\mathbf{E})$, the instantiation $\iota(\mathscr{S},\mathrm{P})$ satisfies the cycle condition $\mathtt{C}_{\iota(\mathscr{S},\mathrm{P})}(\mathbf{E})$.

The proof, which may appear rather technical, essentially holds based on two key ideas:

- the rewriting system properly encodes the cycle property,
- the product of two cycles yields a cycle.

Proof. [139] Let $\mathscr{S} = \mathrm{L} \rightharpoonup \mathrm{R}$ be a $(\mathbb{W} \times \mathbb{D})$-combinatorial rule scheme satisfying the non-orientation condition $\mathtt{O}_{\mathscr{S}}(\mathbf{N})$. Consider the morphisms as in the following diagram, such that vertical spans are products.

$$\begin{array}{ccccc}
\mathrm{L} & \hookleftarrow & \mathrm{L}\cap\mathrm{R} & \hookrightarrow & \mathrm{R} \\
\uparrow{\scriptstyle \rho_{\mathrm{L}}} & & \uparrow{\scriptstyle \rho_{\mathrm{L}\cap\mathrm{R}}} & & \uparrow{\scriptstyle \rho_{\mathrm{R}}} \\
\mathbb{E}_{\mathbb{D}}(\mathrm{P})\times\mathrm{L} & \hookleftarrow & \mathbb{E}_{\mathbb{D}}(\mathrm{P})\times(\mathrm{L}\cap\mathrm{R}) & \hookrightarrow & \mathbb{E}_{\mathbb{D}}(\mathrm{P})\times\mathrm{R} \\
\downarrow{\scriptstyle \tau_{\mathrm{L}}} & & \downarrow{\scriptstyle \tau_{\mathrm{L}\cap\mathrm{R}}} & & \downarrow{\scriptstyle \tau_{\mathrm{R}}} \\
\mathbb{E}_{\mathbb{D}}(\mathrm{P}) & & \mathbb{E}_{\mathbb{D}}(\mathrm{P}) & & \mathbb{E}_{\mathbb{D}}(\mathrm{P})
\end{array}$$

▶ *Sub-condition 1 of the cycle condition: optimal paths are strongly coherent.*

Let $p = v_s \rightsquigarrow v_t$ be (i,j)-optimal path in $\iota(\mathrm{L},\mathrm{P})$. And denote $\hat{p} = \hat{v}_s \rightsquigarrow \hat{v}_t$ the corresponding path in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{L}$. Thus, $\hat{v}_s$ and $\hat{v}_t$ are nodes of $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times (\mathrm{L} \cap \mathrm{R})$. In particular, there exist nodes a_s, a_t in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and u_s, u_t in $\mathrm{L} \cap \mathrm{R}$ such that $\tau_{\mathrm{L}\cap\mathrm{R}}(\hat{v}_s) = a_s$, $\rho_{\mathrm{L}\cap\mathrm{R}}(\hat{v}_s) = u_s$, $\tau_{\mathrm{L}\cap\mathrm{R}}(\hat{v}_t) = a_t$ and $\rho_{\mathrm{L}\cap\mathrm{R}}(\hat{v}_t) = u_t$. By construction of the product, $\hat{p}$ yields two paths $p_{\mathrm{P}} = \tau_{\mathrm{L}}(\hat{p})$ in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$, and $p_{\mathrm{L}} = \rho_{\mathrm{L}}(\hat{p})$ in L. The first one is (i,j)-alternating in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P}))$, from a_s to a_t. The other is (i,j)-optimal in $\pi_{\mathbb{D}}(\mathrm{L})$, from u_s to u_t. Both p_{P} and p_{L} share the same label as paths from graphs in $(\mathbb{W} \times \mathbb{D})$-**Graph**. Because the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathtt{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$, p_{L} is coherent and there is a path p_{R} from u_s to u_t, (i,j)-optimal in $\pi_{\mathbb{D}}(\mathrm{R})$ such that p_{L} and p_{R} have the same label in $\pi_{\mathbb{D}}(\mathscr{S})$. The coherence in $\mathscr{S}$ ensures that $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p_{\mathrm{L}})) \overset{*}{\leftrightarrow}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}} l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p_{\mathrm{R}}))$. The paths p_{P} and p_{R} create an (i,j)-alternating path $\hat{p}'$ in $\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R}$. The coherence in $\mathscr{S}$ and $\mathrm{P} \models \mathtt{C}_{\mathrm{P}}(\mathbf{N},\mathbf{E})$ ensure that $\hat{p}'$ is a path from $\hat{v}_s$ to $\hat{v}_t$. Let p' be the path $\pi_{\mathbb{D}}(\hat{p}')$ of $\iota(\mathrm{R},\mathrm{P})$ associated with $\hat{p}'$. Since p_{L} is optimal, the path p' only contains added arcs and does not overlap. Besides, this path is maximal (if not, it would be included in a path obtained from R, meaning p_{L} would not be coherent). Similarly, it cannot belong to a cycle, as such a cycle would come from R. Therefore, p' is optimal in $\iota(\mathrm{R},\mathrm{P})$ and coherent with p. The incident arcs condition on the combinatorial rule and the incident arcs constraint on the pattern graph ensures the uniqueness of the optimal paths, which are, therefore, strongly coherent.

The reverse proof holds by symmetry, and the scheme instantiation $\iota(\mathscr{S},\mathrm{P})$ satisfies sub-condition 1 of Definition 40.

▶ *Sub-condition 2 of the cycle condition.*

Let v be a preserved node of $\iota(\mathrm{L}\cap\mathrm{R},\mathrm{P})$ that is the source of an i-arc in $\iota(\mathrm{L}\setminus\mathrm{R},\mathrm{P})$. Therefore, there exist a node a in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and a node u in $\mathrm{L}\cap\mathrm{R}$ such that $\tau_{\mathrm{L}\cap\mathrm{R}}(v) = a$ and $\rho_{\mathrm{L}\cap\mathrm{R}}(v) = u$. By construction of the product, the existence of an i-arc of source v in $\iota(\mathrm{L}\setminus\mathrm{R},\mathrm{P})$ means there is an i-arc in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P}))$ of source a and an i-arc in $\pi_{\mathbb{D}}(\mathrm{L})$ of source u. Since the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathtt{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$, the i-arc in R either belongs to an $ijij$-cycle or to an (i,j)-optimal path.

- Assume that the i-arc belongs to an (i,j)-optimal path. Because P satisfies $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$, the (i,j)-optimal path in $\pi_{\mathbb{D}}(\mathrm{R})$ can be associated with an (i,j)-alternating path of source a in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$. Similar to the proof for the coherence of optimal paths (proof of sub-condition 1), the paths in $\mathbb{E}_{\mathbb{D}}(\mathrm{P})$ and R yield an (i,j)-optimal path in $\iota(\mathrm{R},\mathrm{P})$.

- Otherwise, the i-arc extends to an $ijij$-cycle. The concatenated word w in $\pi_{\mathbb{W}}(\mathrm{R})$, corresponding to the cycle, can be reduced to ϵ using $\mathrm{R}_{\mathbf{N},\mathbf{E}}$. Because P satisfies $\mathtt{C}_{\mathrm{P}}(\mathbf{N},\mathbf{E})$, w labels

a cycle in P and thus in $\pi_{\mathbb{W}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P}))$. The product construction keeps this cycle and $\pi_{\mathbb{D}}(v)$ is the source of an $ijij$-cycle in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(\mathrm{P}) \times \mathrm{R})$.

Thus, the scheme instantiation $\iota(\mathscr{S}, \mathrm{P})$ satisfy sub-condition 2 of Definition 40.

▶ *Sub-condition 3 of the cycle condition.*
The proof is similar to the previous sub-condition.

Thereafter, $\iota(\mathscr{S}, \mathrm{P})$ satisfies $\mathtt{C}_{\iota(\mathscr{S},\mathrm{P})}(\mathrm{E})$. □

Now that we have lifted all conditions from rules to rule schemes, we can discuss the transformation of Gmaps and Omaps.

4.7 Consistency preservation for the combinatorial models

Recall that an n-Gmap is a totally labeled $0..n$-topological graph satisfying $\mathtt{O}_{\mathrm{G}}(0..n)$, $\mathtt{I}_{\mathrm{G}}(0..n)$, and $\mathtt{C}_{\mathrm{G}}(0..n_{+2})$ where $0..n_{+2}$ denotes the dimensions $\{(i,j) \in (0..n)^2 \mid i+2 \leq j\}$. From Theorems 1, 2, and 3 (see Chapter 3), if, for a rule r, we have $r \models \mathtt{O}_r(0..n)$, $r \models \mathtt{I}_r(0..n)$, and $r \models \mathtt{C}_r(0..n_{+2})$, then the result graph H of any direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ is an n-Gmap. Similarly, if $r \models \mathtt{O}_r(2..n)$, $r \models \mathtt{I}_r(1..n)$, and $r \models \mathtt{C}_r(1..n_{+2})$ then any graph H result of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ where G is an n-Omap is also an n-Omap.

Graph transformations built on production rules satisfying the appropriate conditions yield derivations preserving the topological constraints of the considered model. When extending rules to rule schemes, the verifications are lifted to the rule scheme level (Theorems 4, 5, and 6). In particular, the result from Theorem 4 ensures the dangling condition can be checked independently of the combinatorial graph on which the rule scheme is applied.

Gmaps and Omaps are defined by slightly different constraints, namely the orientation of the 1-arcs for the Omaps. These constraints yield different conditions for the preservation of the topological consistency when applying rule-based graph transformations. When transposing the conditions to rule schemes, we need to cope with dimension conjugation. In the model of Gmaps, the whole set $\mathbb{D}$ of dimensions is non-oriented, such that $\mathbf{N} = \mathbb{D}$ and $\mathbf{O} = \emptyset$. With $\mathbf{N}$ equals to $\mathbb{D}$, the construction of rule schemes is drastically simplified, and we can do away with dimension conjugation. In the case of Omaps, $\mathbb{D}$ is split into $\mathbf{N} = \mathbb{D} \setminus \{1\}$ and $\mathbf{O} = \{1\}$ meaning that we actually need to take care of the conjugation condition from Theorem 6.

When applying a rule scheme to an n-Gmap or n-Omap, the pattern graph built during the application process inherits properties from the combinatorial graph. For instance, consider an n-Gmap G. Thus, any pattern graph P built on G with a set of words $\mathbb{W}$ satisfies the consistency constraint for pattern graphs $\mathtt{I}_{\mathrm{P}}(\mathbb{W})$, $\mathtt{O}_{\mathrm{P}}(\mathbb{W})$, and $\mathtt{C}_{\mathrm{P}}(0..n, 0..n_{+2})$. Therefore, the instantiation of any rule scheme $\mathscr{S}$ such that $r \models \mathtt{I}_{\mathscr{S}}(0..n)$, $r \models \mathtt{O}_{\mathscr{S}}(0..n)$, and $r \models \mathtt{C}_{\mathscr{S}}(0..n_{+2})$ yields a rule $\iota(\mathscr{S}, \mathrm{P})$ that satisfies the conditions $\mathtt{O}_{\iota(\mathscr{S},\mathrm{P})}(0..n)$, $\mathtt{I}_{\iota(\mathscr{S},\mathrm{P})}(0..n)$ and $\mathtt{C}_{\iota}(\mathscr{S}, \mathrm{P})0..n_{+2}$. Therefore, any direct transformation on any instantiation of the rule scheme results in an n-Gmap. Similarly, rule schemes satisfying the appropriate conditions yield instantiations that transform n-Omaps into n-Omaps. Consequently, we can certify that a rule preserves the model's constraints.

We now discuss some partial implementation of a syntactic analyzer that checks for consistency preservation. As already discussed in Chapter 3, we represent rules with partial monomorphisms. We use names as identifiers for nodes and exploit the combinatorial property of graphs and rules to retrieve the arc function from the node function. Therefore, a rule $L \hookrightarrow R$ consists of two graphs L and R where the preserved nodes have the same name in both sides. For the study of cycles, we need some further constructions. First, let us point out that a function providing all graph cycles can be obtained with a union-find strategy. Start with paths that consist of arcs labeled i and j and unify them recursively whenever one is the source of the other and the dimensions alternate. When the construction stops, the set of paths is pruned to keep only the cycles. Besides, we compute optimal paths for one side of a rule scheme. The function to compute optimal paths is similar to the construction of cycles, only considering non-preserved arcs. In this case, the obtained set of paths should also be pruned to eliminate paths that are sub-paths of cycles.

In the case of Omaps and Gmaps, we have that $\|\mathbf{O}\| \leq 1$ and that for any two pairs (i,j) and (j,k) in $\mathbf{E}$, the pair (i,k) is also in $\mathbf{E}$. Indeed, if $i+2 \leq j$ and $j+2 \leq k$, then $i+2 \leq k$. Therefore, the critical pair analysis reveals that $R_{\mathbf{N},\mathbf{E}}$ is confluent. When a string rewriting system is noetherian and confluent, any word admits a unique normal form, which can be computed in polynomial time by left-most derivation [23, Chap. 2]. Besides, the congruence of words can be tested by checking that both words have the same normal form. We write $\mathcal{NF}_{R_{\mathbf{N},\mathbf{E}}}(w)$ for the normal form a word w in $\overline{\mathbb{D}}^*$, for the rewriting system $R_{\mathbf{N},\mathbf{E}}$.

The verification should first run Algorithm 1, then Algorithm 2 and Algorithm 3 as the last two algorithms assume that the input rule scheme satisfies the incident arcs condition. The algorithms only check for one dimension (resp. pair of dimensions for the cycle condition) and, therefore, should be run on all dimensions relevant to the model.

In Algorithm 1, we check the condition $\mathtt{I}_{\mathscr{S}}(i)$ for a rule scheme $\mathscr{S}$ and a dimension i from Definition 49. Lines 6 to 10 ensure that a preserved node is the source (resp. target) of an i-arc in the left-hand side if and only if it is the source (resp. target) of an i-arc in the right-hand side. Lines 13 to 17 ascertain that each added node is the source (resp. target) of an i-arc, i.e., $\pi_{\mathbb{D}}(R)$ satisfies $\mathtt{I}_{(V_R \setminus V_L),\pi_{\mathbb{D}}(R)}(i)$. Similarly, line 12 certifies that each deleted node is the source (resp. target) of an i-arc, i.e., $\pi_{\mathbb{D}}(L)$ satisfies $\mathtt{I}_{(V_L \setminus V_R),\pi_{\mathbb{D}}(L)}(i)$. This last sub-condition yields the gluing condition on the instantiated rule, which guarantees applicability (provided that an instantiation is possible).

In Algorithm 2, we check the condition $\mathtt{O}_{\mathscr{S}}(i)$ for a rule scheme $\mathscr{S}$ and a dimension i from Definition 51. This condition boils down to only having oriented arcs in the interface, where the definition of reverse arc is extended to encompass the conjugation of the $\mathbb{W}$ part of the label. We construct the set of possible reverse arcs in line 4. The first three boolean conditions $s_L(e') = t_L(e)$, $t_L(e') = s_L(e)$, and $l_{\pi_{\mathbb{D}}(L)}(e') = i$ ensure that the arcs are reverse arcs in the core $\pi_{\mathbb{D}}(\mathscr{S})$. These boolean conditions tackle the first sub-condition of Definition 51. The last boolean condition states that the reverse arcs in the core have conjugate labels, according to the second sub-condition of Definition 51. When we find one, we make sure that both have the same status (deleted or preserved) at lines 5 to 10. Otherwise, i.e., when the arc is oriented, we check whether it is preserved at line 12.

In Algorithm 3, we check the condition $\mathtt{C}_{\mathscr{S}}(i,j)$ for a rule scheme $\mathscr{S}$ and a pair of dimensions (i,j) from Definition 55. The condition is obtained from the condition for instantiated rules on

the scheme's core and an extension with the rewriting system. The sub-condition on the core (see Definition 40) is verified by lines 8 (path coherence), lines 24 to 27 and 30 to 32 (preserved nodes), lines 29 to 31 (added nodes). The extension with the rewriting system to coherence with $R_{\mathbf{N},\mathbf{E}}$ is covered by lines 10 to 15 (path coherence) and lines 17 to 20 (cycles reduction to ϵ).

The algorithms are similar to Jerboa's syntactic analyzer [12]. Jerboa [12] is a generator of geometric modelers based on Gmaps and presented in detail in Chapter 6.

Algorithm 1: Check incident arcs consistency for a dimension i.

Input: A rule scheme $\mathscr{S} = \mathrm{L} \hookrightarrow \mathrm{R}$, and a dimension i of $\mathbb{D}$.
Output: True if the rule satisfies $\mathtt{I}_{\mathscr{S}}(i)$, False otherwise.

1 **Function** *check_incident_arcs(L, R, i)*:
2 consistent $\leftarrow$ True
3 **foreach** $v \in \mathrm{V_L}$ **do**
4 $\mathrm{S_L} \leftarrow \{e \in \mathrm{E_L} \mid s_{\mathrm{L}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{L})}(e) = i\}$
5 $\mathrm{T_L} \leftarrow \{e \in \mathrm{E_L} \mid t_{\mathrm{L}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{L})}(e) = i\}$
6 **if** $v \in \mathrm{V_R}$ **then** // Preserved node
7 consistent $\leftarrow$ consistent **and** $|\mathrm{S_L}| \leq 1$ **and** $|\mathrm{T_L}| \leq 1$ // At most one incident arc
8 $\mathrm{S_R} \leftarrow \{e \in \mathrm{E_R} \mid s_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i\}$
9 $\mathrm{T_R} \leftarrow \{e \in \mathrm{E_R} \mid t_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i\}$
10 consistent $\leftarrow$ consistent **and** $|\mathrm{S_L}| = |\mathrm{S_R}|$ **and** $|\mathrm{T_L}| = |\mathrm{T_R}|$ // Same in both sides
11 **else** // Deleted node
12 consistent $\leftarrow$ consistent **and** $|\mathrm{S_L}| = 1$ **and** $|\mathrm{T_L}| = 1$ // Look for a unique deleted arc
13 **foreach** $v \in \mathrm{V_R}$ **do**
14 **if** $v \notin \mathrm{V_L}$ **then** // Added node
15 $\mathrm{S_R} \leftarrow \{e \in \mathrm{E_R} \mid s_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i\}$
16 $\mathrm{T_R} \leftarrow \{e \in \mathrm{E_R} \mid t_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i\}$
17 consistent $\leftarrow$ consistent **and** $|\mathrm{S_R}| = 1$ **and** $|\mathrm{T_R}| = 1$ // Look for a unique added arc
18 **return** consistent

Algorithm 2: Check non-orientation consistency for a dimension i.

Input: A combinatorial rule scheme $\mathscr{S} = \mathrm{L} \hookrightarrow \mathrm{R}$, and a dimension i of $\mathbb{D}$.
Output: True if the rule satisfies $\mathtt{O}_{\mathscr{S}}(i)$, False otherwise.

1 **Function** *check_non_orientation(L, R, i)*:
2 consistent $\leftarrow$ True
3 **foreach** $e \in \{e' \in \mathrm{E_L} \mid l_{\pi_{\mathbb{D}}(\mathrm{L})}(e) = i\}$ **do**
4 $\mathrm{I_L} \leftarrow \{e' \in \mathrm{E_L} \mid s_{\mathrm{L}}(e') = t_{\mathrm{L}}(e)$ **and** $t_{\mathrm{L}}(e') = s_{\mathrm{L}}(e)$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{L})}(e') = i$ **and** $l_{\pi_{\mathbb{W}(\mathrm{L})}}(e') = \overline{l_{\pi_{\mathbb{W}(\mathrm{L})}}(e)}\}$
5 **if** $|\mathrm{I_L}| = 1$ **then** // Found a reverse arc
6 $r \leftarrow e' \in \mathrm{I_L}$
7 **if** $e \in \mathrm{E_R}$ **then** // Preserved arc
8 consistent $\leftarrow$ consistent **and** $r \in \mathrm{E_R}$ // The reverse must be preserved too
9 **else** // Deleted arc
10 consistent $\leftarrow$ consistent **and** $r \notin \mathrm{E_R}$ // The reverse must be deleted too
11 **else** // No reverse arc
12 consistent $\leftarrow$ consistent **and** $e \in \mathrm{E_R}$ // The arc must be preserved
13 (...) idem (lines 3 to 12) for arcs in $\mathrm{E_R}$ by considering functions s_{R}, t_{R}, $l_{\pi_{\mathbb{D}}(\mathrm{R})}$, and $l_{\pi_{\mathbb{W}(\mathrm{R})}}$, as well as switching the roles of $\mathrm{E_L}$ and $\mathrm{E_R}$.
14 **return** consistent

Algorithm 3: Check cycles consistency for a pair of dimensions (i, j).

Input: A combinatorial rule scheme $\mathscr{S} = \mathrm{L} \hookrightarrow \mathrm{R}$, and two dimensions i and j of $\mathbb{D}$.
Output: True if the rule satisfies $\mathtt{C}_{\mathscr{S}}(\{(i, j)\})$, False otherwise.

1 **Function** *check_cycles(L, R, i, j)*:
2 consistent $\leftarrow$ True
3 $\mathrm{Paths_L} \leftarrow \{p = v_s \rightsquigarrow v_t \in \mathrm{L} \mid \mathrm{p}$ is (i, j)-optimal in L$\}$
4 $\mathrm{Paths_R} \leftarrow \{p = v_s \rightsquigarrow v_t \in \mathrm{R} \mid \mathrm{p}$ is (i, j)-optimal in R$\}$ // Optimal paths
5 $\mathrm{Cycles_L} \leftarrow \{p = v_s \rightsquigarrow v_t \in \mathrm{L} \mid \mathrm{p}$ is an (i, j)-cycle in L$\}$
6 $\mathrm{Cycles_R} \leftarrow \{p = v_s \rightsquigarrow v_t \in \mathrm{R} \mid \mathrm{p}$ is an (i, j)-cycle in R$\}$ // Cycles
// Check paths coherence
7 **foreach** $p \in \mathrm{Paths_L}$ **do**
8 $\mathrm{C} \leftarrow \{p' \in \mathrm{Paths_R} \mid s_{\mathrm{L}}(p) = s_{\mathrm{R}}(p')$ **and** $t_{\mathrm{L}}(p) = t_{\mathrm{R}}(p')$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{L})}(p) = l_{\pi_{\mathbb{D}}(\mathrm{R})}(p')\}$
9 **if** $|\mathrm{C}| = 1$ **then**
10 $c \leftarrow p' \in \mathrm{C}$
11 $\mathrm{N}_p \leftarrow \mathscr{NF}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(\mathrm{L})}(p))$ // The normal form of the label of p
12 $\mathrm{N}_c \leftarrow \mathscr{NF}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(\mathrm{L})}(c))$
13 consistent $\leftarrow$ consistent **and** $\mathrm{N}_p = \mathrm{N}_c$ // Labels are congruent
14 **else**
15 consistent $\leftarrow$ False
16 (...) idem (lines 5 to 12) for paths in R
// Check cycles
17 **foreach** $p \in \mathrm{Cycles_L}$ **do**
18 consistent $\leftarrow$ consistent **and** $(|p| \in \{2, 4\})$ // Check cycle size
19 $\mathrm{N} \leftarrow \mathscr{NF}_{\mathrm{R}_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(\mathrm{L})}(p))$
20 consistent $\leftarrow$ consistent **and** $\mathrm{N} = \epsilon$ // Reduction to the empty word
21 (...) idem (lines 14 to 17) for cycles in R
// Check extension to path or cycle
22 **foreach** $v \in \mathrm{V_R}$ **do**
23 i_arc $\leftarrow$ null
24 **if** $v \in \mathrm{V_L}$ **then** // Preserved node
25 $\mathrm{S_L} \leftarrow \{e \in \mathrm{E_L} \mid s_{\mathrm{L}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{L})}(e) = i\}$
26 **if** $|\mathrm{S_L}| = 1$ **then**
27 i_arc $\leftarrow e \in \mathrm{E_R}$ such that $s_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i$ // Uniquely exists from Alg. 1
28 **else** // Added node
29 i_arc $\leftarrow e \in \mathrm{E_R}$ such that $s_{\mathrm{R}}(e) = v$ **and** $l_{\pi_{\mathbb{D}}(\mathrm{R})}(e) = i$ // Uniquely exists from Alg. 1
30 $\mathrm{P} \leftarrow \{p \in \mathrm{Cycles_R} \cup \mathrm{Paths_R} \mid \text{i\_arc} \in p\}$ // Empty set if i_arc = null
31 consistent $\leftarrow$ consistent **and** $|\mathrm{P}| \neq 0$
32 (...) idem (lines 20 to 28) for dimension j
33 **return** *consistent*

4.8 Application to the quad subdivision

In Section 4.1, we talked about the quad subdivision operation dedicated to mesh refinement. The rule scheme of Figure 4.26 describes the subdivision operation for Gmaps, while the one of Figure 4.27 illustrates it for Omaps.

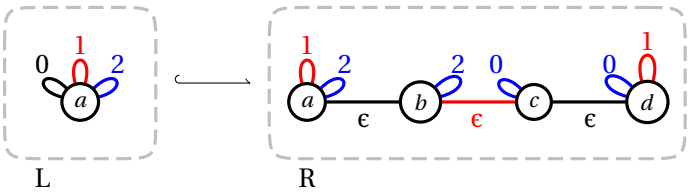


Figure 4.26: Quad subdivision operation: rule scheme $\mathscr{S}_G$ for a 2-Gmap.

Example 61 (Consistency verification of the quad subdivision operation for a 2-Gmap)**.** The rule scheme $\mathscr{S}_G$ of Figure 4.26 is a $(\{\epsilon,0,1,2\},\{0,1,2\})$-rule schemes. It satisfies $\mathtt{I}_{\mathscr{S}_G}(\{0,1,2\})$ since the preserved node a has incident non-oriented 0-, 1-, and 2-arcs in both L and R while added nodes b, c, and d have incident arcs for each dimension in $\{0,1,2\}$. All arcs are non-oriented, and the only arcs with $\mathbb{W}$ label different from ϵ are loops. Thus, $\mathscr{S}_G$ also satisfies $\mathscr{O}_{\mathscr{S}_G}(\{0,1,2\})$. Finally, $\mathscr{S}_G$ also satisfies $\mathtt{C}_{\mathscr{S}_G}(\{(0,2)\})$ because all nodes are sources of 0202-cycles whose $\mathbb{W}$ label reduces to ϵ via $R_{\{0,1,2\},\{(0,2)\}}$. For instance, the cycle $(\epsilon,0)(2,2)(\epsilon,0)(2,2)$ of source a corresponds to a 0202-cycle. The $\{\epsilon,0,1,2\}$-part of the label is $\epsilon 2\epsilon 2 = 22$. Since 2 is non oriented, $(22,\epsilon)$ is a rewriting rule of $R_{\{0,1,2\},\{(0,2)\}}$ and the label reduces to ϵ. The execution of Algorithms 1, 2, and 3 will return True for all relevant dimensions. From these three conditions, we deduce that the rule scheme always transforms a 2-Gmap into a 2-Gmap.

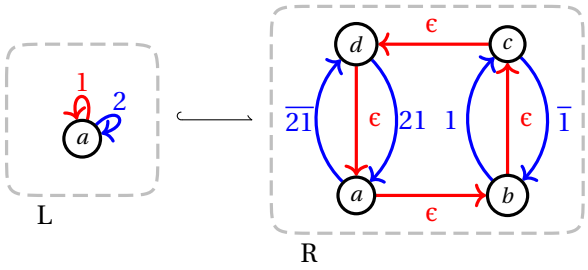


Figure 4.27: Quad subdivision operation: rule scheme $\mathscr{S}_O$ for a 2-Omap.

Example 62 (Consistency verification of the quad subdivision operation for a 2-Omap)**.** The rule scheme $\mathscr{S}_O$ of Figure 4.27 is a $(\{\epsilon,1,\overline{1},2,21,\overline{21}\},\{1,2\})$-rule schemes. The execution of the Algorithms 1, 2 respectively for dimension sets $\{1,2\}$ and $\{2\}$ will return True as $\mathscr{S}_O$ satisfies $\mathtt{I}_{\mathscr{S}_O}(\{1,2\})$ and $\mathtt{O}_{\mathscr{S}_G}(\{2\})$. Note that, as 1 is an oriented dimension, the conjugation part of the condition of non-orientation needs to be considered. The cycle condition does not need verification because a 2-Omap is not subject to any cycle constraint.

Both rule schemes can be instantiated with any 2-Gmap, resp. 2-Omap, allowing subdivision of a connected component within an object. For instance, the subdivisions of the volumes in the cover figure of the chapter (Figure 4.1), the character from Figure 4.10, or the volume in Figure 4.28 can be obtained via the rule schemes of Figure 4.26 or 4.27. All these images were realized with Jerboa.

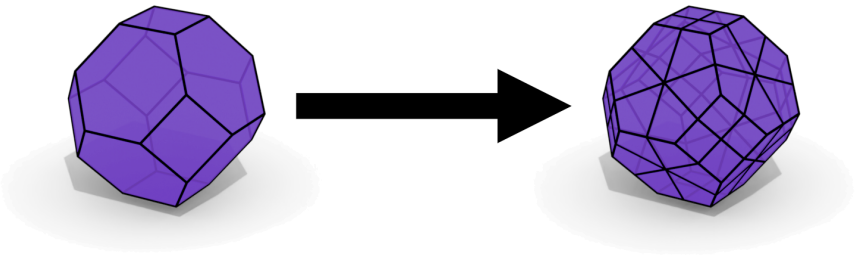


Figure 4.28: Quad subdivision applied to a regular volume.

Example 63 (Rule scheme instantiation for the quad subdivision operation)**.** The left-hand side of $\mathscr{S}_{\mathrm{G}}$ (Figure 4.26) contains a single node and arcs labeled (i,i) for each dimension i in $\{0,1,2\}$. Therefore, the instantiation with any pattern graph obtained via the $\{\epsilon,0,1,2\}$-pattern functor on a 2-Gmap yields a graph equal (up to isomorphism) to the initial 2-Gmap. The rule scheme instantiates into a rule that is always applicable. In practice, the selection mechanism of Section 4.3.4 makes the scheme rule applicable to a connected component. If we consider the character from Figure 4.10 (see Section 4.1), the instantiation left-hand-side of the rule scheme $\mathscr{S}_{\mathrm{G}}$ produces a DPO rule whose left-hand side contains 100680 nodes and 302040 arcs. Since the right-hand side of the scheme rule contains four nodes and 12 arcs (counting the non-oriented arcs as two arcs), the right-hand side of the instantiated DPO rule contains 402720 nodes and 1208160 arcs. In practice, the application of such a scheme rule can be parallelized to speed up the computation time [26]. Likewise, the left-hand side of $\mathscr{S}_{\mathrm{O}}$ (Figure 4.26) contains a single node and arcs labeled (i,i) for each dimension i in $\{1,2\}$. For the same reasons, the instantiation on a pattern graph obtained by the associated functor will always yield a complete connected component in any 2-Omap. Therefore, the instantiated rule is always applicable.

Summary of the chapter's contributions

This chapter provided a product-based generalization of DPO rules called rule schemes. This generalization results in rules parameterized by a set of words. These words represent paths in the underlying combinatorial graph. To modify such paths, we build pattern graphs retrieved unambiguously from a Gmap, resp. an Omap, by the specification of the word set and the mapping of a small set of nodes, called hooks. We lifted the topological consistency conditions of Chapter 3 to rule schemes and provided algorithms to check these conditions.

The missing part to properly represent objects deals with the geometric information added to the topological structure manipulated so far. This missing part is the topic of the next chapter.

Chapter 5

Rewriting objects with geometric embeddings

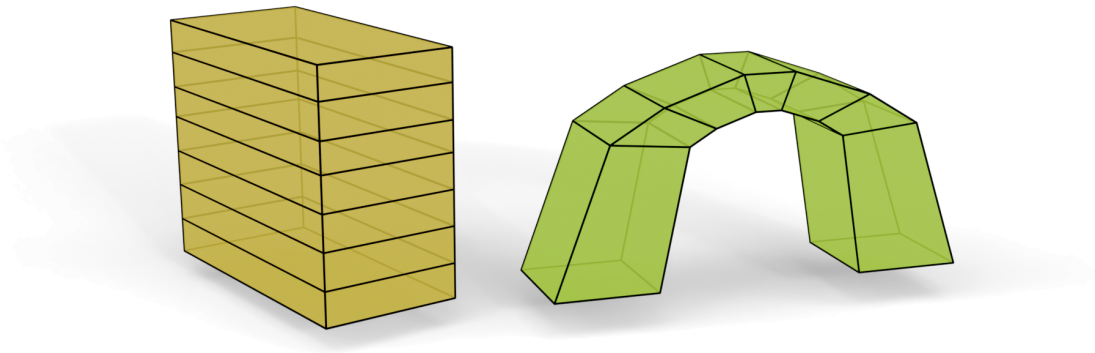


Figure 5.1: These two objects are topologically equivalent but are geometrically distinct. Changing the color and twisting the object on the left yields the arch on the right. Since this transformation does not modify the topology, it only corresponds to a modification of the geometric properties.

Personal note on the chapter

This chapter deals with the representation of geometric information on the object, generically called embeddings. It constitutes works started while I was a Master's student on a subject initiated by Thomas Bellet, which led to a publication in [3].

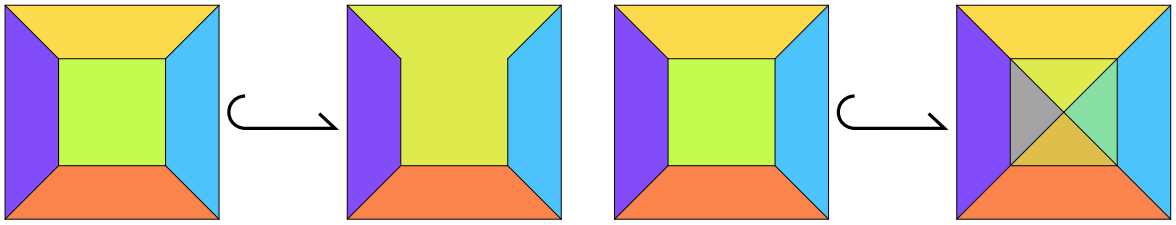
While previous chapters were only of topological content, exploiting the graph's structure to describe an object's subdivision into topological cells, the present one uses decorations on graphs to represent the embeddings. These decorations were handled as labels in [3], and I propose to treat them as attributes in my dissertation. This revised version presents the same contributions as in the publication but in a framework more common within the graph transformation community. I want to emphasize that the main issue related to the embeddings resides in the absence of a suitable graph rewriting theory to describe decorations on graphs that depend on some traversals of the graph structure.

Here are two (maybe relieving) pieces of information for any reader who has followed the presentation so far while struggling with the graph transformation theory. First, the formal part of this dissertation ends with this chapter. Secondly, a practical presentation of the main constructions from this formal part will be given through the explanation of the toolset Jerboa in Chapter 6.

Contents

In Chapter 3, we defined Gmaps and Omaps as graphs labeled on arcs with dimensions and subject to conditions ensuring the well-formedness of the objects. The definitions were of topological content, i.e., the structural relations between the object's cells. In Chapter 1, we explained that the complete description of an object comes from its embedding into a geometric space. This embedding maps each topological cell to an embedding value.

In this chapter, we will see how to handle geometric information on objects and how to modify them. Indeed, modeling operations may alter the geometry on top of modifying the underlying topology. For instance, Figure 5.2a depicts the removal of an edge, and Figure 5.2b the triangulation of a face. Both operations modify a colored 2D object by simultaneously transforming the topological structure and its associated embeddings. The edge removal merges two neighboring faces sharing an edge and mixes the two initial colors in the resulting face. The face triangulation splits a face into triangles and computes the color of the new faces as the mix of the colors from the initial face and their neighboring face. The vertex added by the face triangulation is positioned at the barycenter of the initial face. Were the operations only to modify the topology, we would obtain a two-colored face via the edge removal and a missing position on the vertex added by the face triangulation. Such objects are inconsistent, and the central motivation of this chapter is to provide a framework where such operations can be defined while preserving the consistency of the geometry.

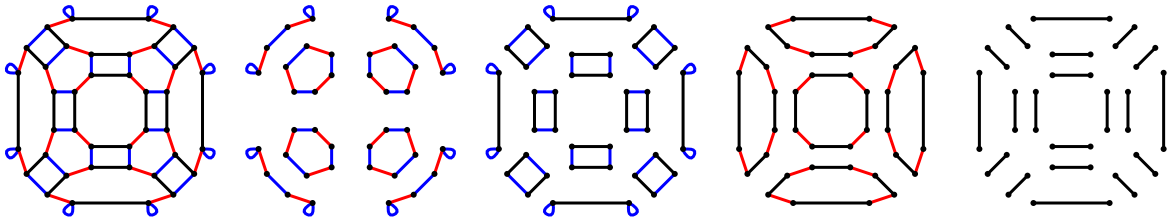


(a) (b)

Figure 5.2: Modeling operations with geometric modifications: (a) edge removal, (b) barycentric triangulation.

Chapter 4 hinted that modeling operations on Omaps modeled as graph transformation rules are more complex than those for Gmaps. Although the only definitional difference appears from the orientation of dimension 1 and the removal of dimension 0, the semantics difference of the graph elements changes the interpretation that we should give to the graph (see the discussion in Section 1.2.2, Chapter 1) Indeed, nodes in a Gmap represent parts of topological vertices, while nodes in an Omap represent parts of topological edges. In Section 3.2.2 of Chapter 3, we saw that topological cells admit different definitions for Gmaps and Omaps. Intuitively, cells correspond to graphs induced by a set of words encoding some involution compositions. In Chapter 4, we used pattern graphs to handle paths. If we restrict the study to the case of Gmaps, the words are all of length 1, i.e., correspond to dimensions. Therefore, we can consider pattern graphs with such one-letter words, which result in the notion of orbits used in [145, 144, 16].

Definition 56 (Orbit)**.** *An* orbit *of a* $\mathbb{D}$*-topological graph* G *consists of a subgraph induced by all the darts reachable from an initial dart, only using links from a subset of* $\mathbb{D}$*. This subset is written* $\langle o \rangle$ *and called an orbit* orbit type*. The orbit is written* $G\langle o \rangle(v)$*, or* $\langle o \rangle(v)$ *when there is no ambiguity on the graph. Such an orbit is said to be of type* $\langle o \rangle$ *or referred to as an* $\langle o \rangle$*-orbit. When a graph* G *is isomorphic to its* $\langle o \rangle$*-orbit, it is called an* orbit graph*.*



(a) (b) (c) (d) (e)

Figure 5.3: Orbits in a 2-Gmap: (a) the 2-Gmap is connected, thus corresponds to its $\langle 0,1,2\rangle$-orbit, (b) the $\langle 1,2\rangle$-orbits corresponding to the topological vertices, (c) the $\langle 0,2\rangle$-orbits corresponding to the topological edges, (d) the $\langle 0,1\rangle$-orbits corresponding to the topological faces, (e) the $\langle 0\rangle$-orbits.

Example 64 (Orbits)**.** The initial object in the operations of Figure 5.2 was used to introduce the topological cells in Chapter 3. The topological structure represented by a Gmap is given again in Figure 5.3a. The topological cells are given in Figures 5.3b (vertices), 5.3d (edges), and 5.3d (faces). However, orbits can represent subgraphs that are not topological cells. For instance, the $\langle 0\rangle$-orbits of Figure 5.3e correspond to 0-arcs and their sources and targets. Since the Gmap consists of a unique connected component, it corresponds to an orbit with all dimensions, i.e., a $\langle 0,1,2\rangle$-orbit.

However, since nodes of an Omap are parts of edges, retrieving vertices requires traversing paths rather than arcs, meaning that the definition of orbits for the representation of topological cells does not hold. In this chapter, we follow the approach we developed in [3], considering only Gmaps and orbits rather than the complete formalism of Chapter 3. The presented contributions have been published in [3] but are partly reformulated here for the overall coherence of the dissertation. All examples will be given in 2D to ease the representation, although all results hold in any dimension. We consider a dimension n used for discussions and definitions. We further assume that $\mathbb{D} = 0..n$.

5.1 Graph attribution

In Chapter 3, we defined topological graphs as arc-labeled graphs. Labels offer a solution to add information to a graph chosen from a given set. Labels do not inherently support operations on the data set used for decoration. Indeed, we might want to decorate the rules with expressions that should be evaluated when the rule is applied to compute actual values. In these cases, attributes constitute the suitable notion of decorations to be added to the graph. In the general case [63], attributes can be used to decorate nodes and arcs. Since we will not require arc attribution, we only present node attribution, i.e., the framework introduced in [98].

5.1.1 Data types

Data types describe data structures in terms of syntax and semantics [62, 184]. The syntax is determined by a signature describing the notations that can be used to refer to the operations available on the data, i.e., the function names. Functions names inductively define terms from a set of variables describing the syntactically valid expressions. The semantics of the data type is given by sets of values for the data and a mapping of the function names to functions on the set of values. The

evaluation of terms within an algebra is defined by the canonical extension of a mapping from the variables to some data values.

Definition 57 (Signature)**.** *A* data type signature $\Omega = (S, F)$ *consists of a set* S *of* type names *and a set* F *of* function names *equipped with a profile mapping* $\rho\colon F \to S^* \times S$*. A function name* f *provided with a profile* $\rho(f) = (s_1, \ldots, s_m, s)$ *is denoted* $f\colon s_1 \times \ldots \times s_m \to s$*. A function with a profile* $(s_1, \ldots, s_m, s)$ *is said to be of* arity m*.*

In the literature, e.g., [62, 184], the type names are also called *sorts* and function names are also called *functions symbols*. Note that the function names with a profile s with $s \in S$ are the constant names of arity 0.

The semantics of the terms is described via an algebra. An algebra consists of a carrier set for each type in the signature, describing the elements in the data type, and a function for each function name, describing the meaning of the symbol. Algebra morphisms are then defined by mapping the carrier sets while preserving the operations.

Definition 58 (Algebra)**.** *Given a signature* $\Omega = (S, F)$*, an* Ω-algebra $\mathscr{A}$ *consists of a family of non-empty carrier sets* $(\mathscr{A}_s)_{s\in S}$ *and a function* $f^{\mathscr{A}}\colon \mathscr{A}_{s_1} \times \ldots \times \mathscr{A}_{s_m} \to \mathscr{A}_s$ *for each function name* f *with profile* $\rho(f) = (s_1, \ldots, s_m, s)$*.*

A morphism of Ω*-algebra* $g\colon \mathscr{A} \to \mathscr{B}$ *is a family of maps* $(g_s\colon \mathscr{A}_s \to \mathscr{B}_s)_{s\in S}$ *such that for all function names* $f \in F$ *with profile* $\rho(f) = (s_1, \ldots, s_m, s)$ *and all typed elements* $a_1 \in \mathscr{A}_{s_1}, \ldots, a_m \in \mathscr{A}_{s_m}$*,*

$$g_s(f^{\mathscr{A}}(a_1, \ldots, a_n)) = f^{\mathscr{B}}(g_{s_1}(a_1), \ldots, g_{s_m}(a_m)).$$

The Ω*-algebras and morphisms of* Ω*-algebra forms a category* $\mathbf{Alg}(\Omega)$*.*

Adding variables to a signature allows the construction of terms, i.e., well-formed strings using variables and function names. These terms describe well-formed expressions that can be used for computation.

Definition 59 (Terms)**.** *Let* $\Omega = (S, F)$ *be a signature and let* X *be the disjoint union of an* S*-indexed family of sets of* variables $(X_s)_{s\in S}$*, assumed disjoint from* F*. For a type* $s \in S$*, the set* $T_\Omega(X)_s$ *of* terms *of type* s *over* Ω *with variables in* X *is the least set satisfying:*

- *for all variables* x *in* X_s*,* x *is a term of* $T_\Omega(X)_s$*;*
- *for all function names* f *of profile* $\rho(f) = (s_1, \ldots, s_m, s)$ *in* F*, for all terms* $t_1 \in T_\Omega(X)_{s_1}$*,* ...*,* $t_m \in T_\Omega(X)_{s_m}$*,* $f(t_1, \ldots, t_m)$ *is a term of* $T_\Omega(X)_s$*.*

We write $t : s$ *for a term* t *in* $T_\Omega(X)_s$ *and* $\mathrm{Var}(t)$ *for the set of variables in* t*. The set* $T_\Omega(X) = \bigsqcup_{s\in S} T_\Omega(X)_s$ *constitutes the set of terms over* Ω *with variables in* X*.*

Terms over a signature define an algebra called the term algebra.

Definition 60 (Term algebra)**.** *The* Ω-term algebra $\mathscr{T}_\Omega(X)$ *over the set of variables* X *maps each expression to its string representation. The carrier sets of* $\mathscr{T}_\Omega(X)$ *are the sets* $T_\Omega(X)_s$ *of terms of type* s *over* Ω*. For a function name* f *with profile* $\rho(f) = (s_1, \ldots, s_m, s)$*, and terms* $t_1 \in T_\Omega(X)_{s_1}$*,* ...*,* $t_m \in T_\Omega(X)_{s_m}$*,* $f^{\mathscr{T}_\Omega(X)}(t_1, \ldots, t_n)$ *is the string* $f(t_1, \ldots, t_n)$*.*

The term algebra plays a key role since it extends any function $f : X \to \mathscr{A}$ where $\mathscr{A}$ is an Ω-algebra to a unique algebra morphism $f^* : \mathscr{T}_\Omega(X) \to \mathscr{A}$ corresponding to the evaluation of terms.

We will also use the final algebra for typing attributed graphs.

Definition 61 (Final algebra [179])**.** *The category* **Alg**(Ω) *of* Ω*-algebras admits a terminal object* $1_{\mathbf{Alg}(\Omega)}$ *called the* final Ω-algebra*. This algebra has carrier sets reduced to singleton sets and, therefore, only trivial functions.*

Example 65 (Data types for positions and colors)**.** In this chapter, we will represent objects with position and color information via the signature $\Omega(pos, col) = (S(pos, col), F(pos, col))$. The signature consists of the type names $S(pos, col) = \{\texttt{point2D}, \texttt{vector2D}, \texttt{colorRGB}\}$ and the function names $F(pos, col) = \{\texttt{plus}, \texttt{midpoint}, \texttt{blend}\}$. The type names respectively correspond to 2D positions, 2D vectors, and colors. The function names have the following profiles:

- $\texttt{plus} : \texttt{point2D} \times \texttt{vector2D} \to \texttt{point2D}$,
- $\texttt{midpoint} : \texttt{point2D} \times \texttt{point2D} \to \texttt{point2D}$,
- $\texttt{blend} : \texttt{colorRGB} \times \texttt{colorRGB} \to \texttt{colorRGB}$.

The associated algebra $\mathscr{A}(pos, col)$ has the following carrier sets and functions:

- $\mathscr{A}(pos, col)_{\texttt{point2D}} = \mathbb{R}^2$,
- $\mathscr{A}(pos, col)_{\texttt{vector2D}} = \mathbb{R}^2$,
- $\mathscr{A}(pos, col)_{\texttt{colorRGB}} = [0, 1]^3$,
- $\texttt{plus}^{\mathscr{A}(pos,col)}$ corresponds to the function $+$, representing the translation of a point by a vector,
- $\texttt{midpoint}^{\mathscr{A}(pos,col)}$ corresponds to the function $midpoint$, computing the middle point of a line segment defined by its two endpoints,
- $\texttt{blend}^{\mathscr{A}(pos,col)}$ corresponds to the function $blend$, defining the average color of two given colors.

In the rest of the dissertation, signatures introduced by the user will be called *user signatures*. As stated above, for the particular case of the example running throughout the chapter, the user signature will contain the user types `point2D`, `vector2D`, and `colorRGB`.

5.1.2 Attributed graphs

Graphs can now be enhanced with an algebra to handle operations on the associated data. This construction is called graph attribution. Presentations in [99, Chap. 1-2] or [111] provide accessible explanations and constructions of attributed graphs and their transformations. In particular, [99, Chap. 1] distinguishes two main constructions of graph attribution. The most straightforward one equips the graph with functions. Each function corresponds to an attribute and assigns values to the graph nodes. These functions are typically considered partial to allow for undefined

attribute values. However, this approach to attribution does not interact well with other constructions in graph transformation, mostly typed graphs. The other approach extends graphs with the addition of data nodes. Each data node represents one value from the data set. The data value is added to a standard node through an attribute arc, meaning that a modification of the value means changing the target of the attribute arc.

Essentially, the first solution can be seen as more pragmatic, narrowing the construction to the minimal needs for a given application. For instance, [141] used attribution functions to design the graph programming Language GP. The values are stored as labels on the graph, where the labeling set is extended with some structure to handle variables and algebraic operations on the data properly. We essentially used a similar approach in our construction of the node variables and embedding expressions [3], following works started in [15, 14].

A complete description of (typed) attributed graphs may be found in [57, Chap. 8] and we also refer the reader to Section 5.2 of [81] for a concise description of attributed graphs and their properties. The end of the current section recalls the construction and main results of attributed graphs and their transformation.

The category of attributed graphs

We only consider attribution of nodes, similar to the construction in [98], but use the more common construction relying on E-graphs [63]. We present here a curated version of E-graphs without arc attribution. Therefore, our definition diverges from standard textbooks, i.e., [57, Chap. 8], which should be considered for the more general approach.

An E-graph is an extension of a graph with two kinds of nodes and two kinds of arcs.

Definition 62 (E-graphs (revisited from [57]))**.** *An* E-graph $G = (V_G, D_G, E_G, A_G, s_G, t_G, sa_G, ta_G)$ *consists of sets of nodes* V_G*, arcs* E_G*, data nodes* D_G*, and attribution arcs* A_G*. Besides the source and target functions* $s_G, t_G \colon E_G \to V_G$ *for arcs, an E-graph has a source function* $sa_G \colon A_G \to V_G$ *and a target function* $ta_G \colon A_G \to D_G$ *for attribution arcs.*

A morphism of E-graphs $m \colon G \to H$ *consists of functions* $m_V \colon V_G \to V_H$*,* $m_D \colon D_G \to D_H$*,* $m_E \colon E_G \to E_H$*, and* $m_A \colon A_G \to A_H$ *that commutes with the source and target functions of both arcs and attribution arcs.*

E-graphs and their morphisms form the category **EGraph**.

Similar to the notation of graphs, the subscripts $_G$ may be omitted or used without fully specifying the graph when there is no ambiguity. In other words, we might write $G = (V, D, E, A, s, t, sa, ta)$ or write V_H after only defining the graph as H. Concisely, an E-graph admits the following diagrammatic definition, which is an extension of the diagrammatic definition of graphs (the purple part corresponds to the diagram 2.2 given in Chapter 2).

$$E \underset{t}{\overset{s}{\rightrightarrows}} V \xleftarrow{sa} A \xrightarrow{ta} D \tag{5.1}$$

Example 66 (Egraph)**.** The graph in Figure 5.4 describes an E-graph with $V = \{a, b, c, d, e\}$ and $D = \{4, 8, 15, 16, 23, 42\}$. The vertices are drawn as circles, and the data vertices as rounded rectangles. The arcs (in E) are drawn in black, while the attribution arcs (in A) are purple and dashed.

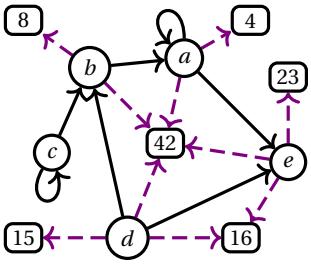


Figure 5.4: An E-graph.

The notion of E-graphs allows building attributed graphs by considering the set of data nodes at the disjoint union of the carrier set of an algebra.

Definition 63 (Attributed graphs [57])**.** *Given a signature* $\Omega = (S, F)$*, an* Ω-attributed graph $AG = (G, \mathscr{A})$ *consists of an E-graph* $G = (V_G, D_G, E_G, A_G, s_G, t_G, sa_G, ta_G)$ *and an* Ω*-algebra* $\mathscr{A}$ *such that* $D_G = \bigsqcup_{s \in S} \mathscr{A}_s$.

A morphism of Ω*-attributed graphs* $m\colon (G, \mathscr{A}) \to (H, \mathscr{B})$ *is a pair* $(m_G, m_{\mathscr{A}})$ *where* $m_G\colon G \to H$ *is a morphism of E-graphs and* $m_{\mathscr{A}}\colon \mathscr{A} \to \mathscr{B}$ *is a morphism of algebras such that the following diagram commutes for all types* s *in* S*, where the vertical monos are inclusions (in* **Set***):*

$$\begin{array}{ccc} \mathscr{A}_s & \xrightarrow{(m_{\mathscr{A}})_s} & \mathscr{B}_s \\ \downarrow & & \downarrow \\ D_G & \xrightarrow[(m_G)_D]{} & D_H \end{array}$$

Ω*-attributed graphs and their morphisms form the category* Ω-**AGraph**.

Example 67 (Attributed graphs)**.** If we consider the signature $\Omega = (\{\texttt{int}\}, \{\texttt{plus}\})$ and the algebra $\mathscr{A}$ with the carrier set $\mathscr{A}_{\texttt{int}} = \mathbb{N}$ and function $\texttt{plus}^{\mathscr{A}} = +_{\mathbb{N}}$, we can extend the graph of Figure 5.4 used in Example 66 to an attributed graph. In this case, we need to consider the set of data nodes $D = \mathbb{N}$. The graphical representation of the graph then exploits the usual convention that only data nodes target of some attribution arc are displayed. For instance, the data node 1960 is not drawn since it is the target of no attribution arc.

Presently, we have no solution to distinguish different kinds of attributes. Similar to the construction in **Graph**, we consider a special graph used to type the attributed graphs. By slicing, we obtain the category of typed attributed graphs. Recall from Chapter 2 that typed graphs can be considered without restriction on the graph used to build the slice category. In the case of attributed graphs, meaningful constructions require considering the final algebra $\mathbb{1}_{\mathbf{Alg}(\Omega)}$ (Definition 61).

Definition 64 (Typed attributed graphs [57])**.** *Given a signature* $\Omega = (S, F)$*, an* Ω-attributed type graph *is an* Ω*-attributed graph* $ATG = (TG, \mathbb{1}_{\mathbf{Alg}(\Omega)})$.

The category Ω-$\mathbf{AGraph}_{ATG}$ *of* typed attributed graphs *over the attributed type graph* ATG *is the slice category* Ω-**AGraph**/ATG.

When we consider an Ω-attributed type graph $ATG = (TG, \mathbb{1}_{\mathbf{Alg}(\Omega)})$, each attribution arc a of A_{TG} represent an attribute. Therefore, attributes can be considered to have a name and a domain value

described respectively by the attribution arc and the type of the target data node in the type graph. In other words, the attributed type graph ATG describes the types and names of the attributes, while a typed attributed graph G stores values from the data type. We will illustrate the notion of attributed type graph in Section 5.2 once we have defined the notion of embedding (see Figures 5.7 and 5.8). Note that the construction of typed attributed graphs allows nodes to have none, one, or several values for a given attribute and none, one, or several attributes.

Since we usually use signatures with a finite set of type names, an attributed type graph contains a finite set of data nodes. However, the algebra of a (typed) attributed graph may have infinite carrier sets. For instance, since integers have an infinite domain, the type of integers has an infinite carrier set. The E-graph part of the (typed) attributed graph stores the disjoint union of the carrier sets, thereby, may be infinite.

Note that a typed attributed graph is a pair G, m where G is an attributed graph and $m\colon G \to$ ATG is a morphism of attributed graphs. However, similar to the consideration for labeled graphs, we will often identify the pair with the graph and talk about the E-graph and the algebra of a typed attributed graph.

Transformation of attributed graphs

The definition of an adequate theory of rewriting relying on the double-pushout approach requires that the category indeed admits suitable constructions, e.g., some pushouts and pullbacks. We seek axioms similar to those of adhesive categories (see Definition 21 in Chapter 2). However, the category Ω-**AGraph**$_{\text{ATG}}$ of typed attributed graphs is not adhesive since it does not admit pushout along all monomorphisms but only along a subclass of monomorphisms [57]. More precisely, the monomorphisms to consider are the morphisms with isomorphisms on the algebra.

Definition 65 (Class $\mathscr{M}$ [57])**.** *Given a signature* $\Omega = (S, F)$*, the class* $\mathscr{M}$ *contains the monomorphisms* $m = (m_G, m_{\mathscr{A}})$ *from* Ω-**AGraph** *where* m_G *is a mono and* $m_{\mathscr{A}}$ *is an isomorphism. We call* $\mathscr{M}$-morphism *the morphisms in* $\mathscr{M}$*.*

In the categories Ω-**AGraph** and Ω-**AGraph**$_{\text{ATG}}$, the following results holds:

- The class $\mathscr{M}$ is a stable system of monos [57, Part III, Chap. 8 and 11], meaning it contains all isomorphisms, it is closed under composition (if f and g are in $\mathscr{M}$, then so is $f \circ g$) and stable under pushouts (see Lemma 5 in Chapter 2).
- The categories have pushouts along $\mathscr{M}$-morphisms, which can be constructed component-wise [57, Part III, Chap. 8].
- The categories have all pullbacks, which can be constructed component-wise [57, Part III, Chap. 8].
- Pushouts along $\mathscr{M}$-morphisms are pullbacks [57, Part III, Chap. 8].
- Pushouts along $\mathscr{M}$-morphisms are van Kampen squares (see Definition 20 in Chapter 2) [57, Part III, Chap. 11].
- The categories are adhesive HLR categories, meaning it has the properties of adhesivity when considering $\mathscr{M}$-morphisms instead of all monomorphisms [57, Part III, Chap. 11].

Adhesive HLR categories are also referred to as $\mathscr{M}$-adhesive categories, which allow distinguishing weak notions of adhesivity such as horizontal/vertical weak $\mathscr{M}$-adhesivity. We do not need these distinctions here since we only need to ensure that pushouts are well-behaved such that DPO rewriting is also well-behaved. Furthermore, attributed graph rewriting usually considers immutable algebras, i.e., rules are spans of attributed graphs where the algebra morphisms are isomorphisms. Intuitively, the rule should not change the rules of computation but only the values associated with the nodes. These are precisely the properties of the class $\mathscr{M}$.

To perform rewriting of (typed) attributed graph, we first choose a category Ω-**AGraph** or Ω-**AGraph**$_{\text{ATG}}$. Choosing the category means choosing the data type signature Ω and, eventually, the attributed type graph ATG. We also choose an Ω-algebra $\mathscr{A}$ describing the set of possible values and how to perform the computations. The rules are constructed with graphs attributed with $\mathscr{T}_\Omega(X)$ the Ω-term algebra with variables in X. More precisely, a rule is a span of $\mathscr{M}$-morphisms where the algebra morphisms are the identity morphism $1_{\mathscr{T}_\Omega(X)}$. The identity morphism on the morphisms of the rule means that the algebra of an attributed graph will be propagated without modification through a direct derivation.

Definition 66 (Attributed rule)**.** *Given a signature* $\Omega = (S, F)$*, an* Ω-attributed rule *r is a span* $L \hookleftarrow K \hookrightarrow R$ *of* $\mathscr{M}$*-morphisms in* Ω-**AGraph** *where the attributed graphs share the algebra* $\mathscr{T}_\Omega(X)$*.*

The construction extends to typed attributed rules *where* L, K, *and* R *are typed over an attributed type graph* $ATG = (TG, 1_{\mathbf{Alg}(\Omega)})$*.*

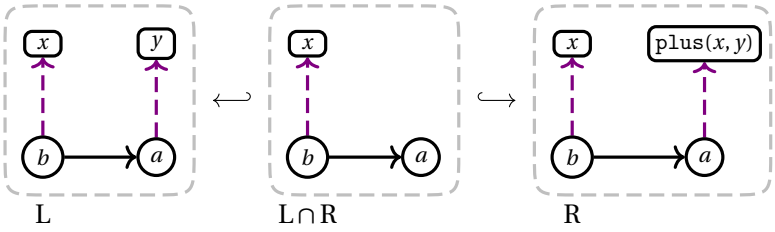


Figure 5.5: An attributed rule.

Example 68 (Attributed rule)**.** The span of Figure 5.5 constitutes an attributed rule. The data nodes are terms describing computations to be realized on the graph. This rule adds the value of the source of an arc to its target via the term `plus`(x, y) attached to the node a.

Given a (type) attributed rule, the notion of direct derivation via DPO rewriting (see Definition 25 in Chapter 2) directly extends from **Graph** and **Graph**$_{\text{TG}}$ to Ω-**AGraph** and Ω-**AGraph**$_{\text{ATG}}$.

Definition 67 (Direct derivation of (typed) attributed graphs)**.** *Let* G *and* H *be two (typed) attributed graphs,* $r = L \hookleftarrow K \hookrightarrow R$ *a (typed) attributed rule, and* $m\colon L \to G$ *a (typed) attributed graph morphism called match. The rule r transforms* G *into* H*, if there is a diagram*

$$\begin{array}{ccccc} L & \hookleftarrow & K & \hookrightarrow & R \\ {\scriptstyle m}\downarrow & (\text{PO}) & \downarrow & (\text{PO}) & \downarrow{\scriptstyle m'} \\ G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

where both squares are pushouts. The direct derivation $G \Rightarrow^{r,m} H$ *exists and is unique (up to isomorphism) if m satisfies the gluing condition, i.e., there exists a pushout complement.*

In the direct derivation $G \Rightarrow^{r,m} H$, the match $m\colon L \hookrightarrow G$ provides the evaluation of the terms appearing in L by mapping the variables to values in G.

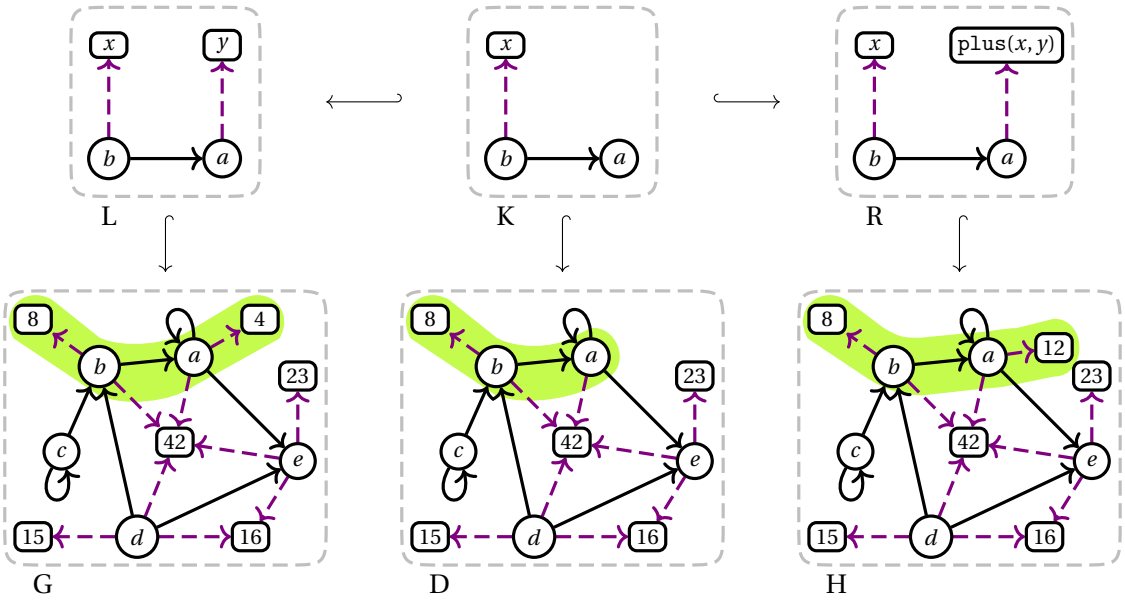


Figure 5.6: A direct derivation of attributed graphs.

Example 69 (Direct derivation of attributed graphs)**.** The Figure 5.6 illustrates a direct derivation of the attributed rule from Example 68 to the attributed graph of Example 67. The match and comatch are highlighted in green. The arc from b to a in the rule's left-hand side is mapped to the arc between b and a in the graph. Nodes a and b are the sources of two attribution arcs. In the occurrence of the match, these attribution arcs are mapped to the attribution arc of target 8 for b and 4 for a. Thus, the algebra morphism maps the variable x onto 8 and the variable y onto 4. Then, the evaluation of the term $\texttt{plus}(x, y)$ yields the value 12. In the result of the direct derivation, the attribution arc of source a and target 8 has been replaced by an attribution arc of source a and target 12.

Note that the existence of a pushout complement can be fully specified in terms of dangling arcs and identifications [57, Part III, Chap. 9]. We will see in Section 5.2 that the treatment of embeddings in topology-based geometric modeling requires additional assumptions. From these assumptions and some restrictions of the general framework, we will derive conditions similar to Fact 1 in Chapter 3, ensuring the existence of a pushout complement.

Before presenting the construction of embeddings, we discuss simplifications of attributed rules and define global variables on rules. A morphism of algebra is described by a set of maps on the carrier sets that preserve the functions of the algebra. Therefore, we should provide such a set of maps when giving a match for a (typed) attributed rule. In practice, we can provide an evaluation of the variables, which canonically extends to an evaluation of the terms corresponding to the needed algebra morphism. Therefore, we further assume some simplifications usual for practical applications, e.g., in [141]. We assume that the terms of the rule's left-hand side are variables.

Assumption 6. *Given a signature* $\Omega = (S, F)$*, a term algebra* $\mathcal{T}_\Omega(X)$ *over the set of variables* X*, and a (typed) attributed rule* $r = L \hookleftarrow K \hookrightarrow R$ *where* L, K*, and* R *share the term algebra* $\mathcal{T}_\Omega(X)$*, we assume that for all all attribution arc* a *in* A_L*, its target* $t_{A_L}(a)$ *is a variable in* X.

Some variables from the terms used in the rule's right-hand side may also appear in the left-hand side. These variables have their value implicitly given by the match. We consider the other variables to be global and parameters of the rule. Recall that for a term t in $T_\Omega(X)$ where Ω is an algebra and X, the set of variables, $\mathrm{Var}(t)$ is the subset of variables from X that appear in t.

Notation 6. *Let* $\Omega = (S, F)$ *be a signature,* $\mathcal{T}_\Omega(X)$ *a term algebra over the set of variables* X*, and* $r = L \hookleftarrow K \hookrightarrow R$ *a (typed) attributed rule where* L, K*, and* R *share the term algebra* $\mathcal{T}_\Omega(X)$*.*

A variable x *in* X *such that there exists an attribution arc* a_R *in* A_R *with* $x \in \mathrm{Var}(t_{a_R}(a_R))$ *but for all attribution arcs* a_L *in* A_L*,* $x \neq t_{a_L}(a_L)$ *is called a* global variable*.*

Global variables are considered parameters of the rules. In the examples, we will write $r(X')$ for a rule where X' constitutes the set of global variables. Such global variables allow computations based on values obtained from input other than the modified graph. For instance, we will use a global variable in Section 5.2.3 to describe the translation of an object vertex by a vector. The vector constitutes the global variable, such that the value of the translation can be specified at execution time.

The subsequent sections describe the addition of embedding functions to Gmaps via node attribution. More precisely, we provide a framework to design embedded modeling operations as attributed graph transformation rules which can be checked for consistency. Like in Chapter 3, we provide constraints on the graphs that define well-formed embedded Gmaps and conditions on the rules to preserve these constraints. In Chapter 4, we extended rules with rule schemes to fulfill the needs of modeling operations. We will again comply with geometric modeling requirements by twisting the rules' application pipeline.

5.2 Embedded Gmaps and their transformations

The construction of the combinatorial models in the previous chapters only covers the topological representation of an object. Geometric information on the topological structure is considered via embedding functions that map topological cells to the relevant data. This section discusses how embedding functions can be represented with attributes through consistency constraints on attributed graphs and how embedded Gmap can be rewritten.

5.2.1 Representation of embedding functions via node attribution

The data is handled via the specification of a signature and an algebra. Before defining embedded Gmaps, we discuss the needs related to the data types in the context of geometric modeling.

Dedicated data types for the embedding representation

In topology-based geometric modeling, each embedding corresponds to a function from topological cells to values from a data type. Topological cells are defined via orbits specified by an orbit type. Therefore, an embedding π is characterized by *embedding function* $\pi : \langle o_\pi \rangle \to \tau_\pi$, where $\langle o_\pi \rangle$ is the orbit type and τ_π is the data type. For an embedding π, the orbit type $\langle o_\pi \rangle$ is called the π*-embedding orbit type*, or simply the *embedding orbit type*. For simplicity, we will refer to an embedding only by its name π. We will then write $\langle o_\pi \rangle$ and τ_π without necessarily specifying $\langle o_\pi \rangle$. We also consider Π as a family of embeddings π.

The definition of embedded Gmaps as attributed graphs starts with a signature and an algebra. Both are partially specified by the set of embeddings Π. Indeed, the set of type names S for the considered signature $\Omega = (S, F)$ should contain all the data types $(\tau_\pi)_{\pi\in\Pi}$. The function names should also describe functions related to the embedding computations. Given a suitable signature Ω, the Ω-algebra contains, as carrier sets, and the set of values for all the data types $(\tau_\pi)_{\pi\in\Pi}$. Note that additional type names may be present in the signature, leading to additional carrier sets in the algebra whenever additional types are required for computation, even if these types are not used for attribution.

Definition 68 (Embedding signature and embedding algebra)**.** *Given a set of embeddings* Π*, an* embedding signature $\Omega(\Pi) = (S(\Pi), F(\Pi))$ *is a signature such that* $\{\tau_\pi \mid \pi \in \Pi\} \subseteq S(\Pi)$*. An* embedding algebra $\mathscr{A}(\Pi)$ *is an* $\Omega(\Pi)$*-algebra. We write* $\lfloor\tau_\pi\rfloor$ *for the carrier sets of type* τ_π *in* $\mathscr{A}(\Pi)$*.*

The notation $\lfloor\tau_\pi\rfloor$ corresponds to $\mathscr{A}_{\tau_\pi}$ in Definition 58. As already discussed in Chapter 3 (see Section 3.2.2), the topological cells are not atomic elements in a Gmap but correspond to subgraphs induced by a subset of dimensions, i.e., an orbit type. The embedding π associates each orbit of type $\langle o_\pi\rangle$ with a value from $\lfloor\tau_\pi\rfloor$. Since attribution of a subgraph is an ill-defined concept, we add values to the nodes of the graph, hence obtaining an attributed graph. Since node attribution does not fully capture the notion of embedding, we need to ensure three additional properties on the attribution arcs.

In the whole chapter, we consider a set of embeddings Π, an embedding signature $\Omega(\Pi)$ for Π, and an embedding algebra $\mathscr{A}(\Pi)$.

Representing multiple embeddings

A Gmap may have several embeddings with different data domains representing different geometrical information, e.g., positions and colors. Therefore, we consider embedded Gmaps as typed attributed graphs.

Definition 69 (Π-embedding type graph)**.** *The* Π-embedding type graph *is the attributed type graph* $\mathrm{ETG}(\Pi) = (\mathrm{TG}(\Pi), \mathfrak{1}_{\mathbf{Alg}(\Omega)})$ *such that* $\mathrm{TG}(\Pi)$ *is the E-graph:*

- $V_{TG(\Pi)} = \{V\}$ *: a singleton set representing the nodes of the topological structure,*
- $D_{TG(\Pi)} = S$ *: one data node per type in the signature,*
- $E_{TG(\Pi)} = (0..n)$ *: one arc per topological dimension,*
- $A_{TG(\Pi)} = \Pi$ *: one attribution arc per embedding,*
- $s_{TG(\Pi)} : (0..n) \to \{V\}$ *: the arcs are loops with source* V*,*
- $t_{TG(\Pi)} : (0..n) \to \{V\}$ *: the arcs are loops with target* V*,*
- $sa_{TG(\Pi)} : \Pi \to \{V\}$ *: the attribution arcs add attributes to the nodes of the topological structure,*
- $ta_{TG(\Pi)} : \Pi \to S, \pi \mapsto \tau_\pi$ *: only embedding values may be used as attributes.*

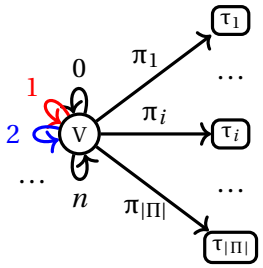


Figure 5.7: The embedding type graph ETG(Π) for a set of embeddings Π.

The graphical representation of ETG(Π) is given in Figure 5.7. Recall that only data nodes targetted by some attribution arc are displayed. In particular, data types used for computations that cannot serve as attributes do not appear in the graph.

In an attributed graph typed over the embedding graph ETG(Π), we will call π-attribution arc any attribution arc mapped to π by the type morphism.

Representing functions via attribution

The usual construction of embeddings relies on functions. Intuitively, the description of embeddings as functions means that each topological cell has an embedding value for each embedding and that the value should be unique. The construction of attributed graphs with E-graphs and data nodes considers attributions with arcs, meaning a node may be the source of none, one of several attribution arcs. The property holds even with the addition of an attributed type graph. Essentially two solutions exist to enforce the existence and uniqueness of an attribution arc for each attribute and each node. We can consider multiplicities [175, 173] on the attributed type graph, similar to the discussion in Section 3.4 (see Chapter 3). In such a case, we add a multiplicity 1 on each attribution arc of the attributed type graph. Such multiplicities may be encoded with one positive and one negative application condition [56]. Keeping the syntax introduced in Section 3.4 of Chapter 3, the positive constraint $\forall (v),\ \exists (v \xrightarrow{\pi_i} \tau_i)$ ensures the existence of an attribute, while the negative constraint $\neg\exists (\tau_i \xleftarrow{\pi_i} v \xrightarrow{\pi_i} \tau_i')$ ensures the uniqueness of the attribute. This construction fits the example in [99, Chap. 4, Section 4.4.1] about the topology of networks for a Voice Over Internet Protocol. The other solution is obtained by analogy with the weak incident arcs condition (Definition 33) and Theorem 1 about the preservation of the incident arcs constraint. The conditions can be stated as follows:

- any added node has a unique attribution arc for each embedding,
- any preserved node has attribution arcs for the same embeddings in the left-hand and right-hand sides.

Then, we obtain a condition similar to the extension from the weak incident arcs condition to the incident arcs condition 34 via a construction similar to Fact 1. The gluing condition can be ensured by imposing that any deleted node is deleted with its attribution arcs for each possible embedding. We prefer this second option, hence obtaining a framework unified with the topology.

Embeddings of orbits

An embedding function π is characterized by its embedding orbit type $\langle o_\pi \rangle$ and a data type τ_π, and viewed as a node attribution. Therefore, we need to ensure that all nodes in the same $\langle o_\pi \rangle$-orbit of a Gmap are attributed with the same value in the carrier set $\lfloor \tau_\pi \rfloor$. This issue constitutes the core of this chapter. We define it as a constraint on the graph and derive a condition on rules to preserve this constraint. Note that the constraint will be defined over subgraphs, meaning that the usual formalism in graph rewriting, i.e., (nested) application conditions, do not provide a suitable framework to express the constraint and preserve it with conditions.

An orbit type $\langle o \rangle$ yields a partition of the nodes of a topological graph since each node belongs to exactly one $\langle o \rangle$-orbit. In other words, an orbit type defines an equivalence relation on the set of nodes.

Definition 70 (Orbit equivalence)**.** *Let* $G = (V, E, s, t, l)$ *be a* $(0..n)$*-topological graph, and* $\langle o \rangle$ *an orbit type with* $\langle o \rangle \in 0..n$*. We consider the relation* $\sim_{\langle o \rangle}$ *on* V *such that for all* e *in* E*, if* $l(e) \in \langle o \rangle$*, then* $s(e) \sim_{\langle o \rangle} t(e)$*.*

The $\langle o \rangle$-equivalence relation $\equiv_{\langle o \rangle}$ *is the reflexive, symmetric, and transitive closure of* $\sim_{\langle o \rangle}$*. We write* $[v]_{\langle o \rangle}$ *for the equivalence class of* $V/\equiv_{\langle o \rangle}$ *containing a node* v*.*

The notion of orbit equivalence yields the well-definedness of an embedding: all nodes in the same $\langle o_\pi \rangle$-equivalence class must have the same value from $\lfloor \tau_\pi \rfloor$. For simplicity, we consider the equivalence relation as being defined by the embedding, meaning that all nodes in the equivalence class are similar when considered from the perspective of the embedding.

Notation 7 (Embedding equivalence)**.** *Given an embedding* $\pi : \langle o_\pi \rangle \to \tau_\pi$*, we write* $\sim_\pi$ *for* $\sim_{\langle o_\pi \rangle}$*,* $\equiv_\pi$ *for* $\equiv_{\langle o_\pi \rangle}$*,* $[v]_\pi$ *for* $[v]_{\langle o_\pi \rangle}$*, and* V/π *for* $V/\equiv_{\langle o_\pi \rangle}$*. Furthermore,* $\equiv_\pi$ *is called the* π-equivalence relation.

Object with position and color embedding

Before detailing the consistency constraints associated with attributes denoting embedding, we illustrate the use of embeddings with E-graphs and (typed) attributed graphs. We consider the addition of colors and positions to objects based on the user signature $\Omega(pos, col)$ containing `point2D`, `vector2D`, and `colorRGB`. Only `point2D` and `colorRGB` are considered for embedding, while `vector2D` is used for computation.

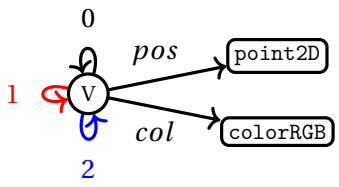
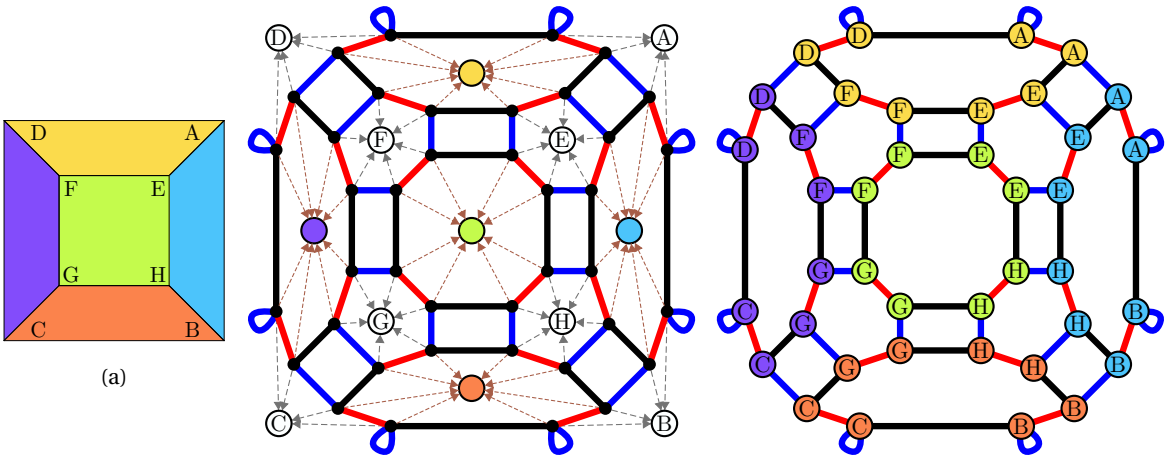


Figure 5.8: The embedding type graph ETG(pos, col) for the position and color embeddings.

Example 70 (Embedding type graph and embedding functions)**.** Figure 5.8 provides the embedding type graph ETG(pos, col) for the position and color embedding defined via the following functions:

- pos: $\langle 1, 2 \rangle \to$ `point2D` associates a 2D position to each vertex.

- col: $\langle 0,1\rangle \to$ `colorRGB` associates a RGB value to each face.



(a) (b) (c)

Figure 5.9: Representation of an embedded object as typed attributed graphs: (a) an object with colored faces and position on vertices, where the positions are given by points in $\mathbb{R}^2$ written as letters, (b) the associated typed attributed graph, (c) an equivalent representation (used for instance in [3]).

Example 71 (A graph embedded with positions and colors)**.** The vertices of Figure 5.9a have positions A, B, C, D, E, F, G, and H. These positions are used as data nodes in the Gmap of Figure 5.9b, where the dashed gray arcs correspond to the pos-attribution arcs.

In Figure 5.9a, the object's faces are colored with ●, ●, ●, ● and ●. These colors yield data nodes in the Gmap of Figure 5.9b where the dashed brown arcs are col-attribution arcs.

In Figure 5.9b, only the E-graph is drawn. The morphism to the embedding type graph and the algebra are missing to describe the typed attributed graph. As in most examples in this chapter, the morphism, type graph, and algebra are left implicit. More precisely, we will consider the graph typed by ETG(pos,col), and the algebra to be $\mathscr{A}(pos,col)$.

Data nodes exist only once per value in a (typed) attributed graph. Therefore, if we were to represent an object with two faces colored ●, the data node ● would appear once. Nodes from both faces would be the source of col-attribution arcs with the same target ●. At this point, graphs tend to be difficult to visualize. Similar to the UML-like notations when graph transformations are used for modeling code and software [99], we display the attributes directly into the nodes like in Figure 5.9c. The topological nodes are colored and decorated with a letter indicating their associated position. We will use this notation throughout the chapter and the second part of this dissertation unless we want to highlight properties related to the attribution arcs. In the case of rules, we write terms inside the nodes. Most examples will be given only position or color embedding to keep the figures readable.

Note that the topological part of the graphs in Figure 5.9b (and 5.9c) are Gmaps, i.e., satisfies the incident arcs constraint (Definition 29) $\mathtt{I}_G(0..n)$, the non-orientation constraint (Definition 30) $\mathtt{O}_G(0..n)$, and the cycle constraint (Definition 31) $\mathtt{C}_G((0..n)_{+2})$. The attribution arcs representing the embedding functions also satisfy previously presented geometric consistency. Indeed, nodes within a $\langle 1,2\rangle$-orbit have the same position, nodes within a $\langle 0,1\rangle$-orbit have the same color, and each node has exactly one position and one color. We will now formalize these constraints.

5.2.2 Consistency constraints for the definition of embedded Gmaps

We now add the geometric data to Gmaps, following works started by Thomas Bellet [15, 14], but using node attribution. The addition of embedding values comes with consistency constraints hinted in Section 5.2.1

Embeded graphs

The embedding constraint describes the properties of an embedding function on orbits in terms of node attribution. The constraint is similar to the embedding constraint defined in [15] and [3] using node labels. As explained in Section 5.2.1, we require that each node is the source of a unique attribution arc per embedding and that all nodes within the same embedding orbit share the same embedding value. These requirements lead to the definition of the embedding constraint$\mathrm{E}_\mathrm{G}(\pi)$ for a graph G and an embedding π. However, we will also consider a weaker form of the constraint, where nodes are only required to be the source of at most one attribution arc (while also keeping the condition on orbits). We call partial embedding constraint this weaker form and write it $\mathrm{E}^{\leqslant}_\mathrm{G}(\pi)$. The partial embedding constraint is used in the rule's interface to allow modification of embedding values; it will also be used in an intermediate step of the rule completion mechanism.

Definition 71 (Embedding constraint)**.** *Let* $\mathrm{G} = (\mathrm{V}, \mathrm{E}, \mathrm{D}, \mathrm{A}, s, t, sa, ta, \mathscr{A}(\Pi), m)$ *be an* $\mathscr{A}(\Pi)$*-attributed graph typed over the* Π*-embedding type graph* $\mathrm{ETG}(\Pi)$ *by the morphism of attributed graph* m*, and* π *be an embedding in* Π*.*

The graph G *satisfies the* partial embedding constraint $\mathrm{E}^{\leqslant}_\mathrm{G}(\pi)$ *if:*

Uniqueness: *every node* v *in* V *is the source of at most one* π*-attribution arc,*

Orbit consistency: *for all nodes* v *and* w *in* V *if* v *and* w *are respectively the source of* π*-attribution arcs* a_v *and* a_w*, and* $v \equiv_\pi w$*, then the attribution arcs have the same target, i.e.,* $ta(a_v) = ta(a_w)$*.*

The partial embedding constraint extends to the embedding constraint $\mathrm{E}_\mathrm{G}(\pi)$ *by replacing the uniqueness constraint by:*

Unique existence: *every node* v *in* V *is the source a unique* π*-attribution arc,*

We write $\mathrm{G} \models \mathrm{E}_\mathrm{G}(\pi)$*, resp.* $\mathrm{G} \models \mathrm{E}^{\leqslant}_\mathrm{G}(\pi)$*, whenever* G *satisfies the embedding, resp. partial embedding, constraint.*

Both constraints can be extended to a subset of embeddings $\Pi' \subseteq \Pi$*. Furthermore, a graph is said to be* partially embedded *by* Π*, resp.* embedded *by* Π*, if it satisfies the partial embedding constraint* $\mathrm{E}^{\leqslant}_\mathrm{G}(\Pi)$*, resp.* $\mathrm{E}_\mathrm{G}(\Pi)$*, i.e., the condition for all embeddings.*

The partial embedding constraint for an embedding π allows $\langle o_\pi \rangle$-orbits to be partially attributed as long as the defined values in $\lfloor \tau_\pi \rfloor$ are equal. For simplicity, we might talk about the (partial) Π-constraint or a (partially) Π-embedded graph to stipulate the embeddings concerned. Whenever a graph satisfies a partial embedding constraint, we can more easily refer to the embedding value associated with a node.

Notation 8. *For a graph* G *satisfying the partial embedding constraint* $\mathrm{E}^{\leqslant}_\mathrm{G}(\Pi)$ *and a node* v *in* V_G*, we write* $\pi(v)$ *for the target value of the* π*-attribution arc of source* v*. We call* π*-value of* v *the*

value $\pi(v)$ *from* $\lfloor \tau_\pi \rfloor$. *Additionally, we write* $\pi(v) = \perp$ *when the node* v *is the source of no arc of type* π. *The notation extends to the equivalence class of the embedding equivalence* $\equiv_\pi$, *in which case we write* $\pi([v]_\pi)$.

Before giving the formalization of embedded Gmaps hinted by the previous definition, we review the construction of typed attributed graphs and rules to find simplifications for our specific needs, i.e., when attributes represent embeddings and when the underlying graph is a Gmap.

The construction of typed attributed graphs and typed attributed rules relied on a series of extensions from standard graphs. Since we are seeking a definition of embedded Gmaps, the series of extensions need to be realized on appropriate graphs. In particular, we strive to keep the notations and constructions introduced in Chapter 3. Therefore, we review the different components of a typed attributed graph to see how they are entangled with previously introduced concepts and perform simplifications whenever possible.

In Chapter 3, we introduced n-Gmaps as graphs with arc labeled with dimensions in $0..n$ subject to three topological constraints. Formally an n-Gmaps corresponds to a graph $G = (V, E, s, t, l)$ where l can be considered as a labeling function on the arcs or as a morphism in **Graph** from $\underline{G} = (V, E, s, t)$ to $\mathbb{1}_{(0..n)}$, the graph with a single node and a loop for each dimension in $0..n$. For a moment, we put aside l and extend $\underline{G}$ to obtain an attributed graph.

Given an embedding signature $\Omega(\Pi)$ and an embedding algebra $\mathscr{A}(\Pi)$, we first need to extend $\underline{G}$ to an E-graph. The extension corresponds to the addition of the data nodes and attribution arcs, as well as the source and target functions for the attribution arcs. From the definition of attributed graphs (Definition 63), we know that the data nodes correspond to the disjoint set of the carrier sets of $\mathscr{A}(\Pi)$. We obtain an E-graph $\hat{G} = (V, D, E, A, s, t, sa, ta)$. We add the algebra $\mathscr{A}(\Pi)$ to $\hat{G}$ resulting in an attributed graph. Finally, we type the pair $(\hat{G}, \mathscr{A}(\Pi))$ over ETG(Π) via a morphism $m = (m_G, m_{\mathscr{A}})$. Note that the algebra of ETG(Π) is the terminal algebra, meaning that $m_{\mathscr{A}}$ is the unique morphism $!_{\mathscr{A}(\Pi)} : \mathscr{A}(\Pi) \to \mathbb{1}_{\mathbf{Alg}(\Omega(\Pi))}$.

The morphism of E-graphs m_G corresponds to function m_V, m_E, m_D, and m_A. The functions m_V and m_E respectively act on the sets of nodes and arcs, such that $(\underline{G}, (m_V, m_E))$ constitute a typed graph (without attribution). The E-graph of ETG(Π), written TG(Π) in Definition 69, as a singleton set for the set of nodes. Therefore, m_V is the trivial function that maps all nodes of V to this singleton set. The function m_E maps the arcs to a dimension in $0..n$ and corresponds to the function l used in Chapter 3. For convenience, we will keep the notation l. The data node function m_D maps each data node to its data type and can be retrieved from the algebra. Finally, the function $m_A : A \to \Pi$ maps each attribution arc to an embedding.

Thereafter, we can uniquely identify an $\mathscr{A}(\Pi)$-attributed graph typed over ETG(Π) from its set of nodes V, edge E, source and target node functions $s, t : E \to V$, arc typing (or label) function l, and node π-values for all embeddings π in Π.

Notation 9. *We write* $G = (V, E, s, t, l, \Pi)$ *for a partially* Π*-embedded graph.*

With the embedding constraint and this lightweight notation, we define embedded Gmaps.

Embedded Gmaps

An embedded Gmap can be constructed as an embedded graph by assuming that the underlying graph satisfies the topological constraints of Chapter 3 defining a Gmap.

Definition 72 (Topological core and embedded Gmaps)**.** *Let* $G = (V, E, s, t, l, \Pi)$ *a* Π*-embedded graph. The* topological core $\mathcal{T}(G) = (V, E, s, t, l)$ *of* G *is the arc-labeled graph of* $(0..n)$-**Graph** *obtained from* G *by removing the data nodes and the attribution arcs. The graph* G *is a* Π-embedded Gmap *if its core* $\mathcal{T}(G)$ *is a Gmap (Definition 32).*

By simplification, we say that an $\mathscr{A}(\Pi)$-attributed graph G typed over the Π-embedding type graph ETG(Π) is a Π-embedded Gmap is it satisfies $\mathtt{I}_G(0..n)$, $\mathtt{O}_G(0..n)$, $\mathtt{C}_G((0..n)_{+2})$, and $\mathtt{E}_G(\Pi)$ without referring to its topological core. When it only satisfies $\mathtt{I}_G(0..n)$ and $\mathtt{E}_G(\Pi)$, we call it a Π*-embedded combinatorial graph.*

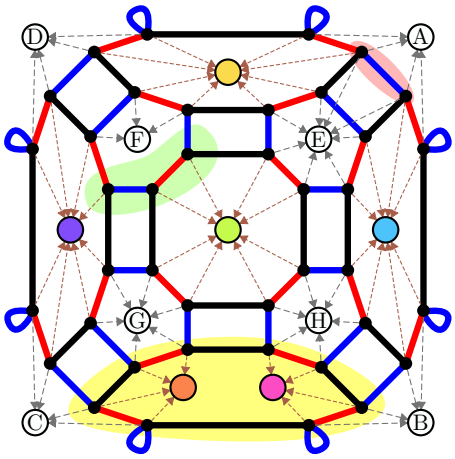


Figure 5.10: A Gmap which is not properly embedded.

Example 72 (Inconsistently embedded Gmap)**.** As discussed previously, the graph of Figure 5.9b is an embedded Gmap. As a comparison, the graph of Figure 5.10 is a Gmap but not an embedded Gmap:

- The nodes in the green area (center left) are not the source of *pos*-attribution arcs (in gray), hence violating the constraint of unique existence.
- The nodes in the red area (top right corner) are the source of two *pos*-attribution arcs, conflicting with the constraint of uniqueness.
- The nodes in the yellow area (bottom) belong to the same $\langle 0, 1\rangle$-orbit, i.e., the same face, which is the orbit type of the color embedding. However, some *col*-attribution arcs target the ● data node and some the ● data node. The nodes describe a bicolor face, which contradicts the constraint of orbit consistency.

We will now provide conditions on rules to preserve the consistency constraints of the embedding functions, similar to the approach developed in Chapter 3.

5.2.3 Modeling operations as transformations of embedded Gmaps

Rules defined in the section introduce the notion of embedded transformations. They constitute the final rules used to modify an embedded Gmap. However, working with these rules would be cumbersome since we would need to define many redundant rules based on the topology of the modified orbit. We choose to work by increment, first discussing a light approach before extending

it to richer rules. This incremental presentation allows for studying the conditions for preserving the embedding constraints in a more straightforward framework.

Recall from Chapter 2 that we consider linear rules where the interface is denoted by L∩R when needed. Intuitively, a rule is a partial monomorphism, which extends to $\mathscr{M}$-morphisms. The rule is indifferently written $L \xhookleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ or $L \rightharpoonup R$.

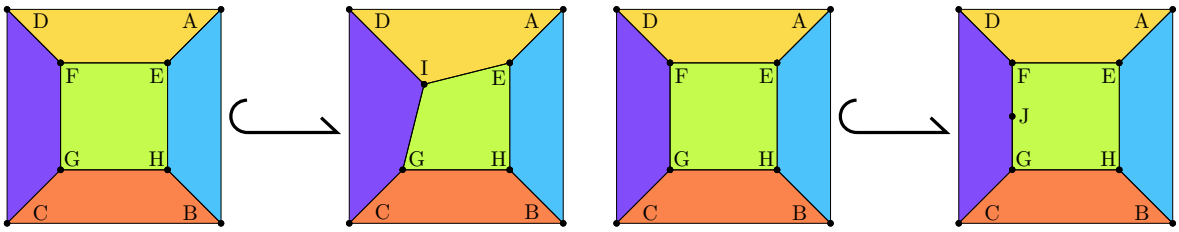


Figure 5.11: Two operations with modifications of the position embedding: (a) vertex translation, and (b) vertex insertion.

We now investigate how modeling operations on embedded Gmaps that modify their geometry can be defined using attributed graph transformation rules according to Definition 66. As an example, we consider the two operations given in Figure 5.11 that act on the position embedding. The vertex translation of Figure 5.11a is a purely geometric operation as it does not affect the topological structure: position F is translated to I. Conversely, the vertex insertion of Figure 5.11b affects both the topological structure and the embedding. An edge of the square face is split into two edges by introducing a new vertex, embedded by the pos-value J.

Match in an embedded graph

The starting point is to consider typed attributed rules $r = L \xhookleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ typed over ETG(Π). Similar to the rules representing the topological operations, a transformation requires the specification of a match from the rule's left-hand side to the modified graph. This match corresponds to a morphism of E-graph and a morphism of algebra. The morphism of algebra maps the terms in the rule's left-hand side to the algebra $\mathscr{A}(\Pi)$ used for embedding the Gmaps. Since a (typed) attributed rule is a span made of $\mathscr{M}$-morphisms where the morphism of algebra is the identity, all vertical morphisms in the DPO diagram will have the same morphism of algebra as the match. Note that Assumption 6 and the use of global variables allow for a graphical representation of the algebra morphism.

We keep the assumptions made in Chapter 3 on the match, extending them to E-graphs. In particular, we assume that the node and arc functions of the E-graph match morphism are injective and characterize any such match as being embedded. We assume no property on the data node and attribution arc functions. Indeed, distinct orbits of the modified embedded graph may have the same value for an embedding. In this case, nodes in distinct orbits will have the same attribute but could be matched from nodes in the rule left-hand side with distinct variables. Such a match is perfectly valid and should not be forbidden. Therefore, the match is not enforced to be a monomorphism. As a final remark, we point out that the injectivity property on the node function of the match with the partial embedding constraint on the left-hand side implied the injectivity of the attribution arc function of the match.

We define embedded morphisms as morphisms satisfying injectivity on the node and arc functions such that an *embedded match* is an embedding morphism used as a match.

Definition 73 (Embedded morphism)**.** *Let* $(m_G, m_{\mathscr{A}})$: $(G, \mathscr{A}) \to (H, \mathscr{B})$ *be a morphism of* $\mathscr{A}(\Pi)$*-attributed graphs typed over the* Π*-embedding type graph* $\mathrm{ETG}(\Pi)$*, and let* m_V: $V_G \to V_H$ *and* m_E: $E_G \to E_H$ *respectively be the node and edge functions of* m_G*.*

The morphism $(m_G, m_{\mathscr{A}})$ *is an* embedded morphism *if* m_V *and* m_E *are injective functions.*

The critical point in defining basic geometric modeling operations is to ensure that the consistency constraints of embedded Gmaps are preserved by rules applications, provided that rules satisfy some syntactic conditions. We gave syntactic conditions for the preservation of topological constraints in Chapter 3. Here, we use a similar approach to investigate syntactic conditions to preserve the embedding constraints, ensuring that embedded Gmaps are transformed into embedded Gmaps.

In an embedded graph, each node is the source of a unique π-attribution arc for each embedding π of Π. This constraint is identical to the incident arcs constraint (Definition 29 in Chapter 3), considering the embeddings instead of the dimensions. Therefore, we can solve the gluing condition like in Fact 1. The gluing condition states the existence of a pushout complement for the match. It can be syntactically checked on the rule by ensuring that any deleted node is deleted with an attribution arc for each embedding. Similar to its topological counterpart, the condition ensures the gluing condition regardless of the match.

Theorem 7 (Gluing condition for embedded graph)**.** *For an attributed rule* $r = L \hookleftarrow R$ *typed over* $\mathrm{ETG}(\Pi)$ *and applied to an embedded Gmap, the gluing condition on a match* $L \to G$ *is equivalent to the following two properties:*

- $\mathscr{T}(L \cap R) \hookrightarrow \mathscr{T}(L) \to \mathscr{T}(G)$ *admits a pushout complement,*
- *any deleted node of* $L \setminus R$ *is the source of a unique* π*-attribution arc for each embedding* π *in* Π*.*

Proof. The proof is the same as Fact 1 given in [144]. □

Rather than dealing with the two constraints of Definition 71 separately as we did with the topological part, we study the preservation as a whole. Indeed, the constraint of the unique existence of embeddings is the same as the incident arcs constraint that we already studied in detail in Chapter 3.

5.2.4 Preservation of the embedding consistency

Before providing the condition and presenting the related theorem, let us illustrate how a transformation may or may not preserve the embedding of orbits constraint. Intuitively, we could consider the rule of Figure 5.12a to describe the vertex translation of Figure 5.11a. The rule has a unique node a, associated with the term x in its left-hand side and the term t in its right-hand side. The term x is a variable used to retrieve the value of the position embedding of the modified node, while the term t stands for $\texttt{plus}(x, \vec{v})$. For readability purposes, we write the terms inside the nodes, the node identifiers (describing the morphisms) outside the nodes, and the complete expressions of the terms above the rule.

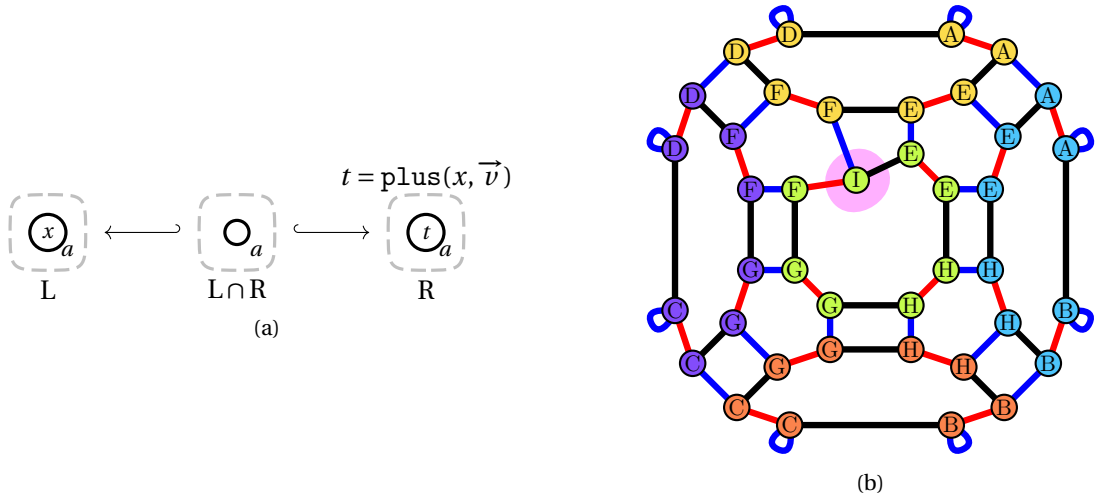


Figure 5.12: Incoherent vertex translation: (a) a rule $r(\vec{v})$ containing only one node, translating its position, (b) an incoherent application, which produces an inconsistently embedded Gmap.

Unfortunately, such a rule is not appropriate for our needs. Indeed, the rule modifies the position of a single node while all nodes in the $\langle 1,2\rangle$ should have their position changed. The result of an application of the rule to the graph of Figure 5.9 is given in Figure 5.12b. The algebra morphism described by the match maps x to I and $\texttt{plus}^{\mathscr{A}(pos,col)}(x, \vec{v})$ to F. The node highlighted in pink has been translated by the vector $\vec{v}$, resulting in the new position I, but the other nodes in its $\langle 1,2\rangle$-orbit still have the position F. The graph of Figure 5.12b is not an embedded Gmap.

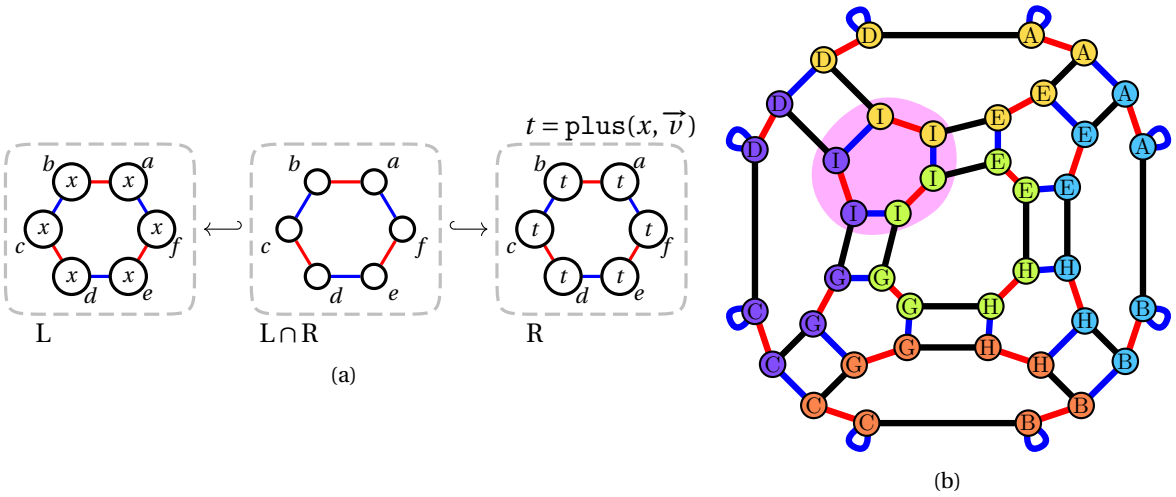


Figure 5.13: Coherent vertex translation: (a) a rule $r(\vec{v})$ containing a complete $\langle 1,2\rangle$-orbit, translating its position, (b) a coherent application, which produces a consistently embedded Gmap.

Embedding values of nodes within an embedding orbit should be modified simultaneously and in the same manner. For example, the rule of Figure 5.13a matches (respectively rewrites) a full vertex orbit in L (respectively in R). All nodes are connected with both 1-arcs and 2-arcs. In fact, both L and R are complete $\langle 1,2\rangle$-orbits. Thus, applying this rule to the embedded Gmap of Figure 5.9 yields the embedded Gmap of Figure 5.13b.

While defining embedded rules, we provide weaker versions that will be used later in the chapter.

Definition 74 (Embedded rule)**.** *Let* $r = \mathrm{L} \rightarrowtail \mathrm{R}$ *be an attributed rule typed over* ETG(Π)*. We consider*

the following conditions on r.

Partially embedded components: L *and* R *are partially embedded graphs, i.e.,* $L \models E^{\leqslant}_{L}(\Pi)$ *and* $R \models E^{\leqslant}_{R}(\Pi)$ *(Definition 71).*

Embedded components: L *and* R *are embedded graphs, i.e.,* $L \models E_{L}(\Pi)$ *and* $R \models E_{R}(\Pi)$ *(Definition 71).*

Full orbits of transformed embeddings: *If* v *is a preserved node of such that* $\pi_L(v) \neq \pi_R(v)$*, then every node of* $R\langle o_\pi\rangle(v)$ *is the source of exactly one* i*-arc in* R *for each* i *of* $\langle o_\pi\rangle$.

The rule r *is a* weakly Π-embedded rule*, or simply a weakly embedded rule if it satisfies the condition of embedded components. The rule is a* partially Π-embedded rule*, or simply a partially embedded rule if it satisfies the condition of partially embedded components and the condition of the full orbits of transformed embeddings. Finally, the rule is a* Π-embedded rule*, or simply an embedded rule if it is both a weakly and partially* Π*-embedded rule, i.e., it satisfies the condition of embedded components and the condition of the full orbits of transformed embeddings. These definitions extend to a set of embeddings* Π.

The condition of embedded components ensures that the interface $L \cap R$ is a partially embedded graph, i.e., $(L \cap R) \models E^{\leqslant}_{L\cap R}(\Pi)$. Indeed, it contains elements both in L and R, hence satisfying the constraint of uniqueness and orbit consistency (Definition 71). However, if a node is the source of different π-attribution arcs in L and R, then it is the source of no such arc in the interface. Thus, $L \cap R$ does not satisfy the constraint of unique existence.

Note that the condition of embedded components partially implies the gluing condition for embedded graphs (Theorem 7). This condition of embedded components is, in a sense, stricter than the incident arcs condition. A weaker condition could be stated as follows:

1. Any preserved node is the source of an π-attribution arc in L if and only if it is the source of an π-attribution arc in R.
2. Any added node is the source of a unique π-attribution arc in R.
3. Any deleted node is the source of a unique π-attribution arc in L.

However, the two conditions are, in fact, equivalent since it suffices to extend this supposedly weaker form with π-attribution arc on all nodes in L and R such that the terms in the right-hand side correspond to the variables used in the left-hand side.

The equivalence between these two considerations for the condition of embedded components upholds the representation of the rule in Figure 5.13a, where the color embedding is not specified. The complete attributed rule (Definition 66) is given in Figure 5.14a. It satisfies the conditions of Definition 74 when considering the position and color embedding. Each node of L and R is the source of a unique pos-attribution arc (in gray) and the source of a unique col-attribution arc (in brown), while nodes of K are the source of a unique col-attribution arc and no pos-attribution arc. The terms c_1, c_2, and c_3 are variables of type `colorRGB` not modified by the rule. The rule consists of a unique $\langle 1,2\rangle$-orbit, and all pos-attribution arcs have the same target. Within each $\langle 0,1\rangle$-orbit, the nodes are the source of col-attribution arcs with the same target.

Thus, the rule satisfies the condition of embedded components. Since only the position is modified and the rule consists of a complete $\langle 1,2\rangle$-orbit, the condition of the full orbits of transformed embeddings is also satisfied. Even though this representation is more explicit, we will mainly use the representation of Figure 5.13a for its compactness. We will also reuse the simplification introduced in Section 3.3.2 in Chapter 3 and write rules as partial monos, i.e., the representation of Figure 5.14b.

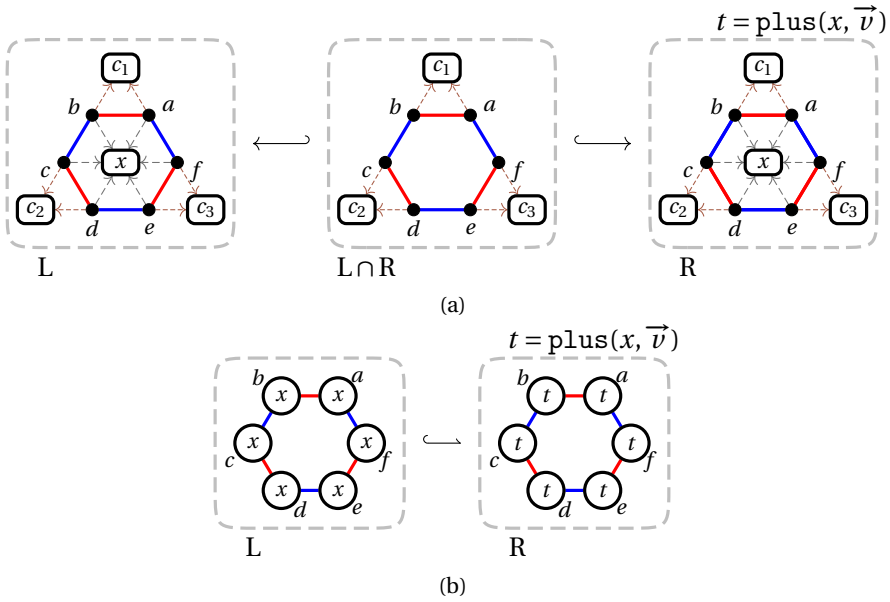


Figure 5.14: Representations of an embedded rule: (a) as a typed attributed rule, (b) as a partial mono.

By considering the core of an embedded rule, we can incorporate topological properties into the rule. We are particularly interested in the preservation of the incident arcs constraint, which enables the consideration of arcs based on dimensions.

Definition 75 (Embedded combinatorial rule)**.** *An embedded rule* $r = \mathrm{L} \hookrightarrow \mathrm{R}$ *is an embedded combinatorial rule if* $\mathcal{T}(r) = \mathcal{T}(\mathrm{L}) \hookrightarrow \mathcal{T}(\mathrm{R})$ *is a combinatorial rule (Definition 35).*

Before giving the expected theorem, i.e., that the embedding of orbits condition preserves the associated constraint, we give counter-examples for the condition.

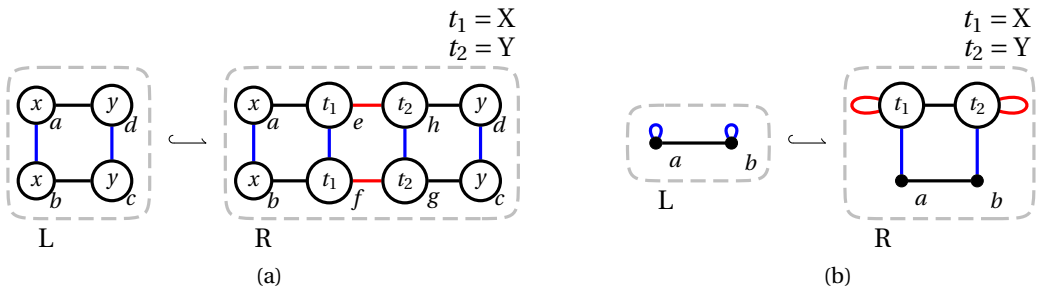


Figure 5.15: Inconsistent rules that break conditions of Definition 74: (a) Inconsistently embedded vertex insertion, (b) Unsafe extension of two vertices. Only the position embedding is considered.

Example 73 (Inconsistent rules)**.** The condition of embedded components states that the rule's left-hand and right-hand sides are embedded graphs, thus encoding two conditions: the orbit consistency and the unique existence (Definition 71). The orbit consistency requires that all

nodes within a $\langle o_\pi \rangle$-orbit share the same π-term. The rule of Figure 5.15a breaks this condition as it adds a new vertex (nodes e, f, g and h) embedded with two different pos-terms t_1 and t_2. The terms correspond to constant function names (of arity 0), meaning the vertex is embedded with two position values X and Y. The constraint of unique existence requires that each node is the source of a unique attribution arc per embedding. In other words, each node must have an embedding term for each embedding. This constraint prevents the extension of orbits by adding nodes or merging orbits. The rule of Figure 5.15b breaks this condition as a half-edge without embedding values is added to another half-edge whose vertices are embedded by the two positions X and Y. Applying such a rule would cause the same inconsistency as the rule from Figure 5.12a: vertices would have two positions. The condition of the full orbits of transformed embeddings ensures that orbits with modified embedding values are full orbits. Thus, all nodes receive the updated value. The rule from Figure 5.12a breaks this condition.

Theorem 8 (Preservation of the embedding consistency)**.** *If* $r = \mathrm{L} \rightarrowtail \mathrm{R}$ *is an embedded combinatorial rule and* $m \colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded combinatorial graph, then the result* H *of the direct transformation* $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ *is an embedded combinatorial graph.*

This theorem highlights the entanglement of the geometry with the topology. Although we could consider the topology as a standalone subject in Chapter 3, the geometry depends on the topology. We request that the modified graph lives in $(0..n)$-**CGraph** to reason about the orbits while only considering properties defined on arcs and nodes. In particular, the condition of the full orbits of transformed embeddings exploits arcs based on dimension, which is made sound by considering combinatorial graphs.

Proof. We only give a sketch of the proof, while the complete version can be found in Appendix B.1. Since a combinatorial rule transforms a combinatorial graph into a combinatorial graph, we only need to prove the two constraints of Definition 71 on H. The first one is the constraint of the unique existence of embedding, stating that each node is the source of a unique attribution arc per embedding. Since R is an embedded graph, all nodes in the comatch of the rule possess a unique value per embedding, while the other nodes keep their unique value from G. The second constraint is the orbit consistency, stating that all nodes in an embedding orbit should have the same embedding value. The key idea to prove this constraint is to reason about an i-arc where i is a dimension in the orbit and exploit the equivalence relation defined by the orbit type of an embedding. We then prove that the source and target of the i-arc have the same embedding value. This part of the proof is done by case analysis. □

Compared to the construction in [3], the use of typed attributed graphs means that we define the conditions for embedded (combinatorial) rules directly on rules with terms. Therefore, the equality of terms within orbits in the graphs of the rule is imposed by the framework.

At this point, we established syntactic conditions on rules that preserve the geometric consistency of an embedded Gmap. However, our defined rules suffer from two significant restrictions.

The first restriction comes from the conditions introduced in this section. The condition of the full orbits of transformed embeddings means that the embedding value associated with an orbit can only be modified if the complete orbit appears in the rule. For instance, the rule of Figure 5.13 explicitly defines the translation of a vertex adjacent to three edges. From a user-end perspective,

operations based on such a specific structure are very restrictive and counter-intuitive. The vertex translation has a single meaning on a semantic level, independent of the number of adjacent edges. A user-friendly rule should be as simple as in Figure 5.16 in which a single node encodes the transformation. The rule matches a node, catches the value of its pos-embedding using a variable, and replaces it with a new position, here x and $x+\overrightarrow{v}$. Nonetheless, the application of this rule can produce an inconsistent object.

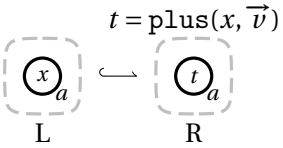


Figure 5.16: Expected scheme of a vertex translation.

The second restriction relates to the encoding of the embedding in the data type, i.e., the signature, the algebra, and the terms. Function names come with a profile and, therefore, an arity, meaning that the number of parameters is fixed. However, operations in geometric modeling usually require accessing multiple values, e.g., all the positions in a face. In other words, operations modifying the geometry of an object may require accessing collections of embedding values. Furthermore, the rules only allow accessing embedding values of elements matched by the rule. This restriction does not hinder the expressiveness of the framework since it suffices to extend the rule with the needed elements to access their embedding value. However, it drastically reduces the practical usability of the rules. For instance, we may want to consider the vertex insertion operation with the new vertex at the midpoint between the barycenter of the two adjacent faces. A similar computation is used in the Catmull-Clark subdivision of surfaces [31], which refines the quad subdivision by geometric smoothing. The subdivision scheme adds vertices to split edges in the middle of the two previously mentioned positions, i.e., at the middle between the edge midpoint and the midpoint between the faces' barycenters. To compute the barycenter of the adjacent faces, we need to add them to the rule. However, we would then have to consider all possible arity of faces, e.g., triangles, quads, and, more generally, all polygons with m edges for all integers m. We then obtain an infinite ruleset.

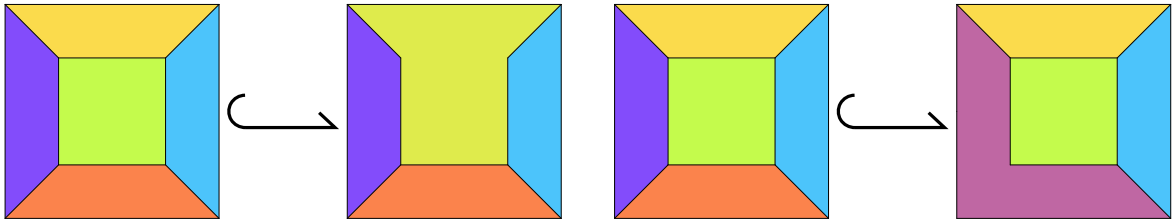
Both issues entangle some topological considerations in the modification of the geometry. The first restriction questions the possibility of modifying embedding values without fully specifying the underlying topology, while the second restriction questions the possibility of accessing it.

5.3 Completion of attributed graph rewriting for geometric modeling

So far, every considered operation has been defined based on the transformed object's specific topological structure. Similar to the approach developed in Chapter 4, we strive to define operations independent of the underlying topology. In this section, we propose a solution to the first restriction previously described, i.e., the modification of embedding values through the topology. The proposed solution is to extend the rewriting mechanism.

5.3.1 Need for simplicity

For instance, the edge removal of Figure 5.17 involves both topological and embedding modifications. On the topological aspect, the edge is removed, and the two adjacent faces are merged. On the embedding aspect, the resulting face's color is obtained by mixing the colors of the two original faces. We use this example to illustrate the topological extension and embedding propagation. For simplicity, we only consider the color embedding.



(a) (b)

Figure 5.17: Edge removals.

Semantically, this operation does not depend on the configurations of the two faces. It should correspond to the simple rule of Figure 5.18a. However, similarly to the translation in Figure 5.12, the rule is inconsistent.

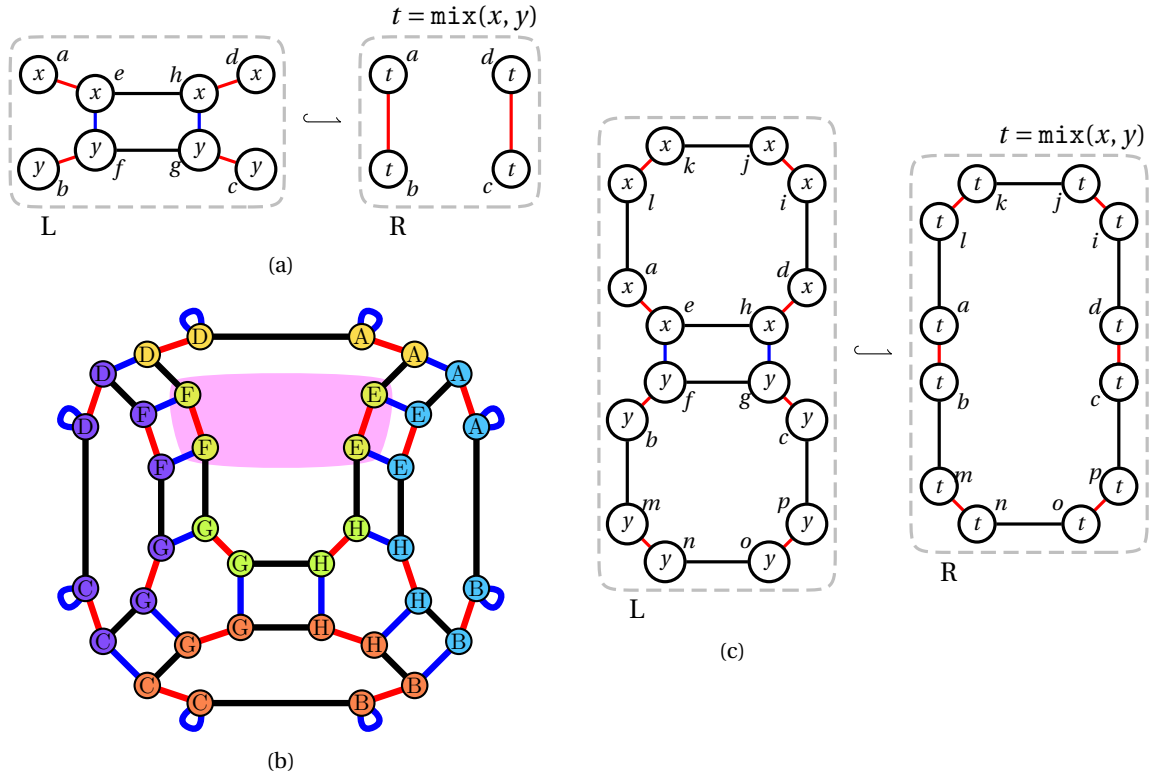


Figure 5.18: Rules for edge removal: (a) intuitive and inconsistent rule, (b) inconsistent application of the rule from Figure (a), and (c) correct(ed) rule.

The application of the rule of Figure 5.18a to the object of Figure 5.9 results in the inconsistent object of Figure 5.18b (where the comatch is signified by the pink area). This inconsistent application yields a face where some nodes are embedded with the color yellow, some with the color green, and others with their mix. The expected behavior is that all nodes should have the mix of

green and yellow as their color. The embedding modifications must be propagated to all nodes of the two faces to preserve the embedding consistency.

Therefore, we derive a mechanism that extends the rule and propagates the embedding modifications. In our example, the rule of Figure 5.18a has to be extended into the correct rule of Figure 5.18c. This extension is realized in two steps: the topological extension that matches all required nodes and the embedding propagation that ensures consistent relabeling.

The complete pipeline is presented in Figure 5.19, introducing the notion of topological extension, denoted by $\oplus m$, and the embedding propagation, denoted by $\odot \pi$.

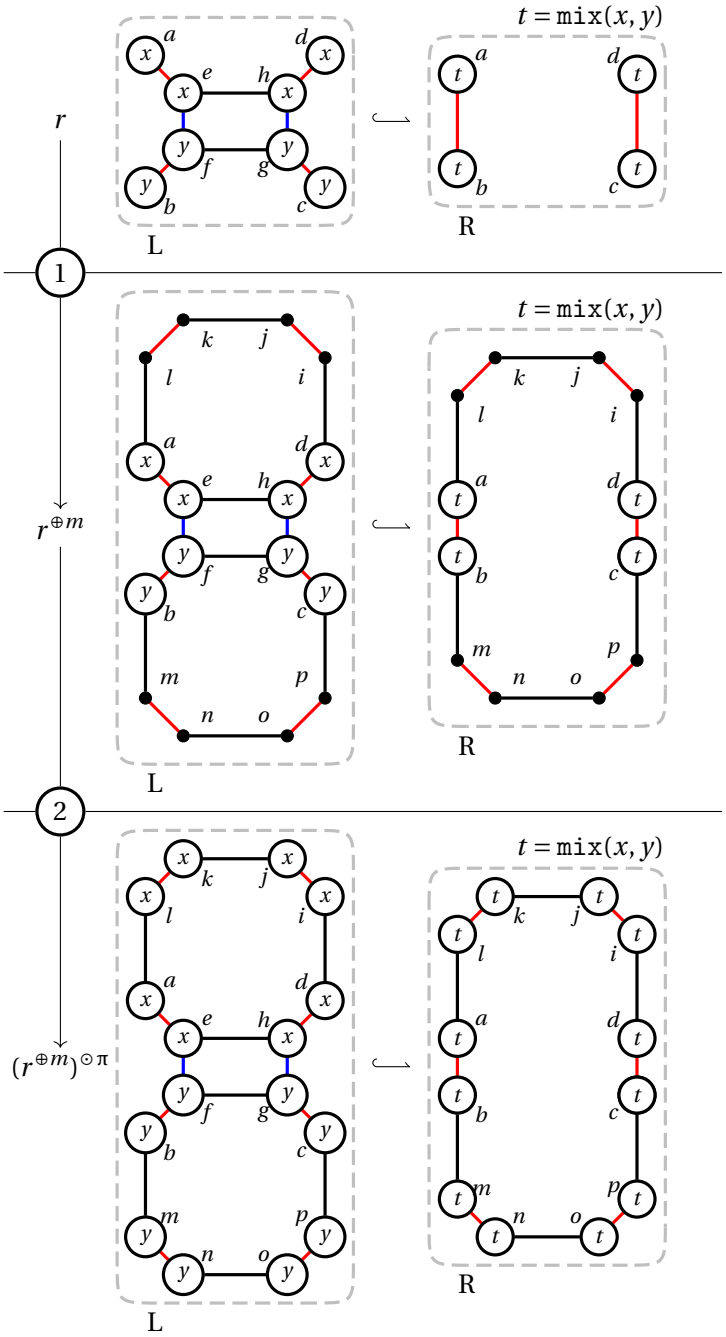


Figure 5.19: Topological extension and embedding propagation.

Since the embedding col is defined on faces (orbit $\langle 0, 1 \rangle$), it is necessary to consider nodes reachable by arcs labeled in $\langle 0, 1 \rangle$. The topological extension, represented in step 1, retrieves the

missing nodes for all $\langle 0,1\rangle$-orbits, extending the match to the yellow and green faces in the original object. These gathered nodes are also added to the interface and the right-hand side of the rule (denoted $r^{\oplus m}$). Note that these nodes are added without embedding (thus drawn as black dots). All nodes added by the topological extension have undefined embedding values. The embedding propagation depicted in step 2 adds embedding terms to the nodes added by the topological extension. We present the topological extension and the embedding propagation without considering the consistency preservation, which is postponed to Section 5.4.

5.3.2 Topological extension

Intuitively, the topological extension (step 2 in Figure 5.19) uses the match morphism to complete the partial embedding orbits with the full orbits of the transformed Gmap. Therefore, we define the orbit completion of a subgraph for an orbit type $\langle o\rangle$ as the smallest graph containing the subgraph and all its $\langle o\rangle$-orbits from the global graph. Recall that $(0..n)$-**Graph** is the category of graphs arc-labeled with dimensions in $0..n$, i.e., the topological part of an embedded Gmap.

Definition 76 (Orbit completion)**.** *Let* G *be a graph in* $(0..n)$-**Graph**, $\mathrm{H} \hookrightarrow \mathrm{G}$ *a subgraph of* G, *and* $\langle o\rangle$ *an orbit type.*

The $\langle o\rangle$-completion *of* H *in* G, *written* $\mathrm{G}\langle o\rangle(\mathrm{H})$, *is the smallest subgraph of* G *containing* H *and all orbits* $\mathrm{G}\langle o\rangle(v)$ *for all nodes* v *in* $\mathrm{V_H}$.

The notation $\mathrm{G}\langle o\rangle(v)$ refers to the orbit of type $\langle o\rangle$ containing the node v while the notation $\mathrm{G}\langle o\rangle(\mathrm{H})$ corresponds to the $\langle o\rangle$-completion of H. Viewing v as a graph reduced to the node v, the $\langle o\rangle$-completion of v coincides with $\mathrm{G}\langle o\rangle(v)$.

The orbit completion is well-defined since $(0..n)$-**Graph** is adhesive, meaning that the category **Sub**(G) of subobjects of G admits coproducts[1] built as pushouts over their product [115]. Furthermore, **Sub**(G) forms a distributive lattice [115], meaning that the orbit completion for several orbit types is well-defined. The products and coproducts yield a unique monomorphism $m_{\langle o\rangle}$: $\mathrm{H} \hookrightarrow \mathrm{G}\langle o\rangle(\mathrm{H})$, given a morphism $m\colon \mathrm{H} \hookrightarrow \mathrm{G}$.

Example 74 (Orbit completion)**.** We identified some nodes in the Gmap G of Figure 5.20a. Node c can be considered a subgraph of G with a unique node. The $\langle 0,1\rangle$-completion of c in Figure 5.20b yields the face incident to c, i.e., $\mathrm{G}\langle 0,1\rangle(c)$, while its $\langle 1,2\rangle$-completion yields the vertex of Figure 5.20c. The color embedding is carried by the face orbit, while the vertex orbit carries the position one. Therefore, the graph of Figure 5.20d contains all nodes necessary to modify the embedding values of c. It corresponds to the union of $\mathrm{G}\langle 0,1\rangle(c)$ and $\mathrm{G}\langle 1,2\rangle(c)$ in **Sub**(G). The face incident to c in Figure 5.20b is a subgraph of G which can also be completed. Its $\langle 1,2\rangle$-completion yields the subgraph $\mathrm{G}\langle 1,2\rangle(\mathrm{G}\langle 0,1\rangle(c))$ of Figure 5.20e. We obtain a different subgraph than in Figure 5.20d since we did sequential completions rather than the union of two completions.

The completion mechanism is well-behaved with the orbit type and subgraph lattices. Indeed, if an orbit type $\langle o'\rangle$ is included in an orbit type $\langle o\rangle$, then the $\langle o'\rangle$-completion of a subgraph is included in its $\langle o\rangle$-completion. For instance, the orbit type $\langle 1\rangle$ is included in both $\langle 0,1\rangle$ and $\langle 1,2\rangle$. Thus, the $\langle 1\rangle$-completion of c in Figure 5.20f is included in both the $\langle 0,1\rangle$-completion of c

[1] Coproduct is the the dual notion of product obtained from Definition 9 by reversing the arrows.

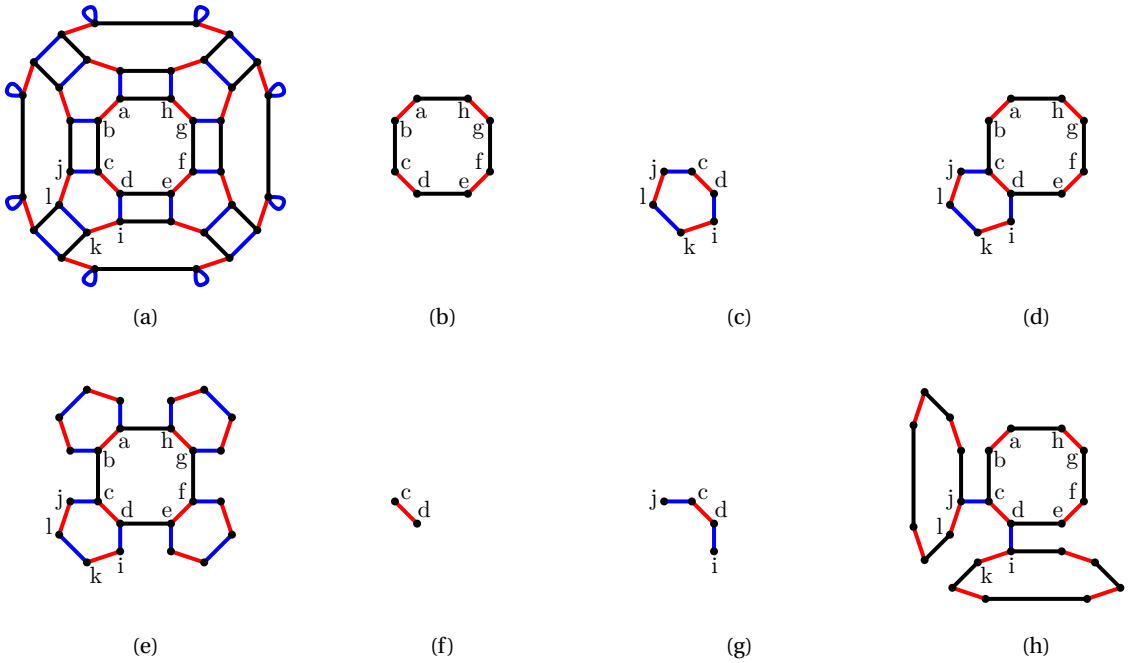


Figure 5.20: Orbit completions: (a) the 2-Gmap G from Figure 5.3 with identified nodes, (b) the $\langle 0,1\rangle$-completion of graph coincident with node c, (c) the $\langle 1,2\rangle$-completion of graph coincident with node c, (d) the union of graphs from Figures (b) and (c), (e) the $\langle 1,2\rangle$-completion of $G\langle 0,1\rangle(c)$, (f) the $\langle 1\rangle$-completion of graph coincident with node c, (g) a subgraph H of G, and (h) the $\langle 0,1\rangle$-completion of H.

(Figure 5.20b) and its $\langle 1,2\rangle$-completion (Figure 5.20b).

The orbit completion is not restricted to graphs that are themselves orbits. The graph H of Figure 5.20g contains one 1-arc and two 2-arcs but its is not a $\langle 1,2\rangle$-orbit of G, since nodes i and j are not the source of 1-arc. The $\langle 0,1\rangle$-completion of H is provided in Figure 5.20h. Since c is a node of H, the lattice property of **Sub**(G) ensure that $G\langle 0,1\rangle(c)$ is included in $G\langle 0,1\rangle(H)$.

We naturally extend the construction of orbit completion to E-graphs and (typed) attributed graphs by considering that only the topological part of the graph is extended. Therefore, the algebra of a (typed) attributed graph is not modified by the orbit completion nor its E-graph's attribution arcs and data nodes. We obtain the Π-completion by considering all the embedding orbit types $\langle o_\pi\rangle$ for π in Π.

Definition 77 (Morphism of embedding completion)**.** *Let* G *and* H *be two attributed graphs typed over the embedding type graph* ETG(Π) *such that* $\mathcal{T}(H) \hookrightarrow \mathcal{T}(G)$*, i.e., the topological core of* H *is a subgraph of the topological core of* G*.*

The morphism of Π-completion *of* H *in* G*, is the morphism* $H \to (G\langle o_\pi\rangle)_{\pi\in\Pi}(H)$ *obtained from the* $\langle o_\pi\rangle$*-completion of* H *for all embeddings* π *in* Π*. The associated algebra morphism is the identity morphism on the algebra of* H*.*

The injectivity of the topological part and the construction of orbit completions ensures the well-definedness of the morphism of embedding completion. Once again, the completion mechanism is only of topological content. Nodes in $(G\langle o_\pi\rangle)_{\pi\in\Pi}(H)$ added by the completion are not embedded. Besides, the completion of embedded morphisms does not modify the algebra morphism.

Since the morphism of embedding completion consists of an identity algebra morphism and a monomorphism of E-graphs, the morphism of embedding completion belongs to $\mathscr{M}$. Besides,

for an embedded morphism $m\colon \mathrm{H} \to \mathrm{G}$ (Definition 73), the injectivity of the node and arc functions ensures that $\mathscr{T}(\mathrm{H}) \hookrightarrow \mathscr{T}(\mathrm{G})$. Therefore, m can be factorized into the morphism m_Π of Π-completion of H in G and a morphism $m'\colon (\mathrm{G}\langle o_\pi\rangle)_{\pi\in\Pi}(\mathrm{H}) \to \mathrm{H}$. In other words, if $m\colon \mathrm{H} \to \mathrm{G}$ is an embedded morphism, there exists a unique embedded morphism $m'\colon (\mathrm{G}\langle o_\pi\rangle)_{\pi\in\Pi}(\mathrm{H}) \to \mathrm{H}$ such that the following diagram commutes.

$$\begin{array}{ccc} \mathrm{H} & \xrightarrow{m_\Pi} & (\mathrm{G}\langle o_\pi\rangle)_{\pi\in\Pi}(\mathrm{H}) \\ & \searrow^{m} & \vdots\,\exists! m' \\ & & \mathrm{G} \end{array} \tag{5.2}$$

The hypothesis for considering the morphism of embedding completion concerns the topological core of the graphs. G and H can have different algebras, data nodes, and attribution arcs. In particular, we can consider G as a graph to be modified and H as the left-hand side of a rule. In such a case, the data nodes of G store embedding values while those of H store terms. The topological extension is obtained via the embedded match. We get the Π-completion of the rule's left-hand side from the modified graph. We then apply the initial rule to the completed graph. The construction of the DPO diagram yields a span called the topological extension of the rule.

Definition 78 (Topological extension)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* ETG(Π) *and* $m\colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded combinatorial graph.*

The topological extension *of* r *along the match* m *is the rule* $r^{\oplus m} = \mathrm{L}^{\oplus m} \hookleftarrow (\mathrm{L}\cap\mathrm{R})^{\oplus m} \hookrightarrow \mathrm{R}^{\oplus m}$ *defined by the following double pushout diagram:*

$$\begin{array}{ccccc} \mathrm{L} & \hookleftarrow & \mathrm{K} & \hookrightarrow & \mathrm{R} \\ {\scriptstyle m_\Pi}\downarrow & & \downarrow & & \downarrow \\ \mathrm{L}^{\oplus m} = (\mathrm{G}\langle o_\pi\rangle)_{\pi\in\Pi}(\mathrm{L}) & \hookleftarrow & \mathrm{K}^{\oplus m} & \hookrightarrow & \mathrm{R}^{\oplus m} \end{array}$$

where m_Π *is the morphism of* Π*-completion of* L *in* G.

The topological extension may not exist, in particular, if $(\mathrm{L}\cap\mathrm{R}) \hookrightarrow \mathrm{L} \hookrightarrow \mathrm{L}^{\oplus m}$ does not admit a pushout complement, i.e., if m_Π for not satisfies the gluing condition. However, since $(\mathrm{G}\langle o_\pi\rangle)_{\pi\in\Pi}(\mathrm{L})$ is a subgraph of G, m_Π satisfies the gluing condition whenever m does. In particular, the gluing condition is implied by the condition of embedded components of embedded rules (Definition 74). The existence of m' in the diagram 5.2 is primordial as it ensures that we have a match $\mathrm{L}^{\oplus m} \to \mathrm{G}$.

Example 75 (Topological extension)**.** The morphism from the graph L in Figure 5.21b to the graph G in Figure 5.21a via the morphism induced by the node names yields the morphism of *col*-completion m_{col} in Figure 5.21b. The graph G of Figure 5.21a is represented without embedding values since they do not change the result of the completion. The application of the rule depicted in the top span of Figure 5.21c yields the topological extension of the bottom span. We write as black dots the added non-embedded nodes and as white nodes the already existing nodes without embedded terms in $\mathrm{L}\cap\mathrm{R}$. In the extended rule, the $\langle 0,1\rangle$-orbits which support the color embedding are complete.

A carefully observant reader could argue that we only need to perform the orbit completion of nodes with modified embedding values. Indeed, completing the orbits of unmodified nodes con-

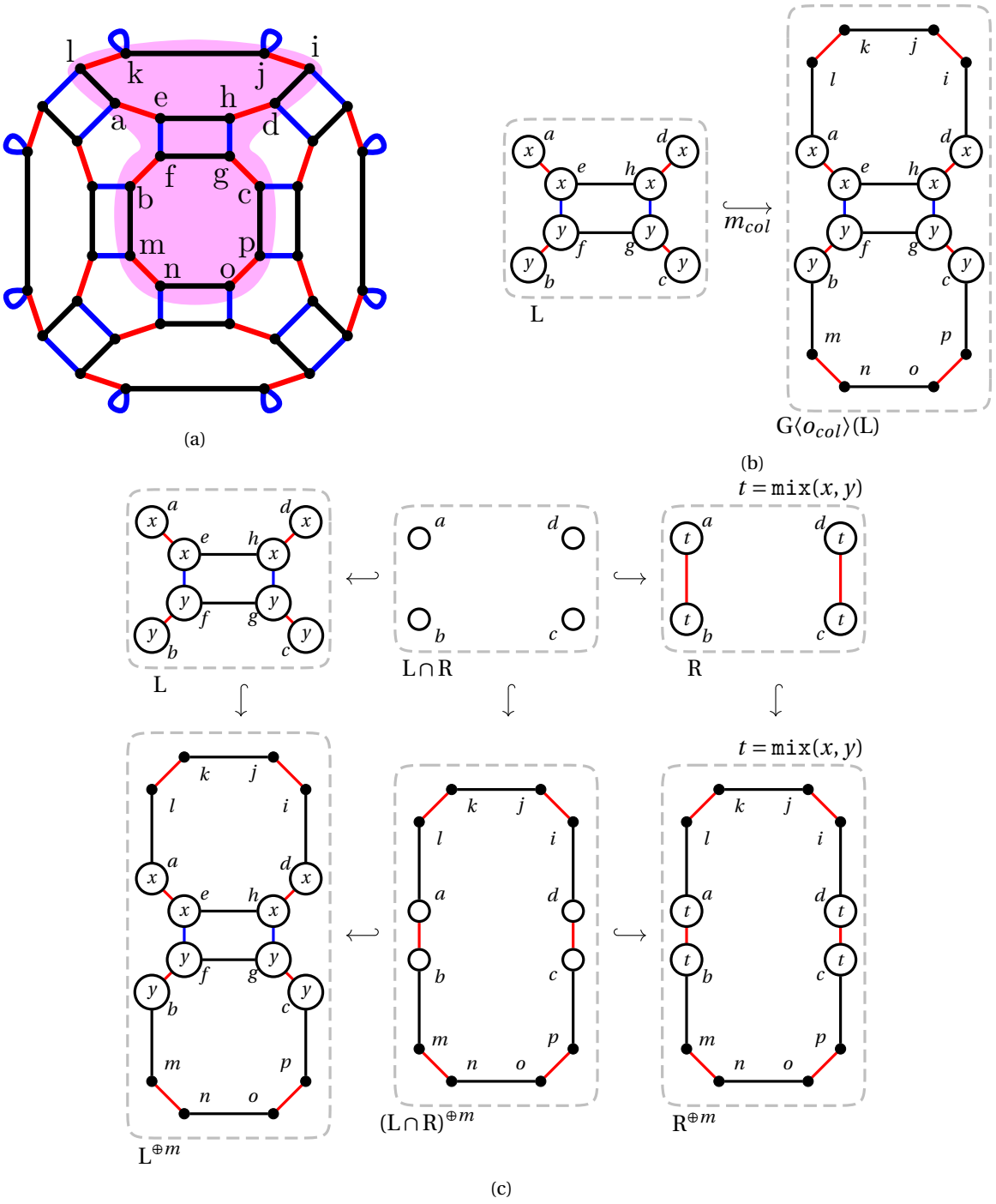


Figure 5.21: Construction of the topological extension: (a) graph used for the topological extension, and (b) morphism of *col*-completion of L in the graph G of Figure (a), (c) topological extension along the match L → G.

tributes nothing to the modification of the graph. However, as already discussed in Section 5.2.4, nodes with the same variable in L and R do not change the effect of the rule. Therefore, the completion does not hinder the expressivity of the rules but simplifies the process.

The topological extension retrieves orbits that were not encoded in the rule but needed to be accounted for to obtain rules preserving the embedding consistency. However, only elements from the topological structure have been added, meaning that the extended rule breaks the conditions of Definition 74. Indeed, the rule's left and right-hand sides satisfy the uniqueness constraint and

not the unique existence constraint: nodes added by the extension do not have embedding values. We propagate the embedding terms on the extended rule to obtain consistent rules.

5.3.3 Embedding propagation

In the extended rule of Figure 5.21c, embedding terms have to be propagated in order to obtain the final rule of Figure 5.22. The embedding propagation constitutes a direct application of the orbit equivalence (Definition 70). For all graphs of the extended rule, each node is embedded with the embedding term of its equivalence class (assuming one exists).

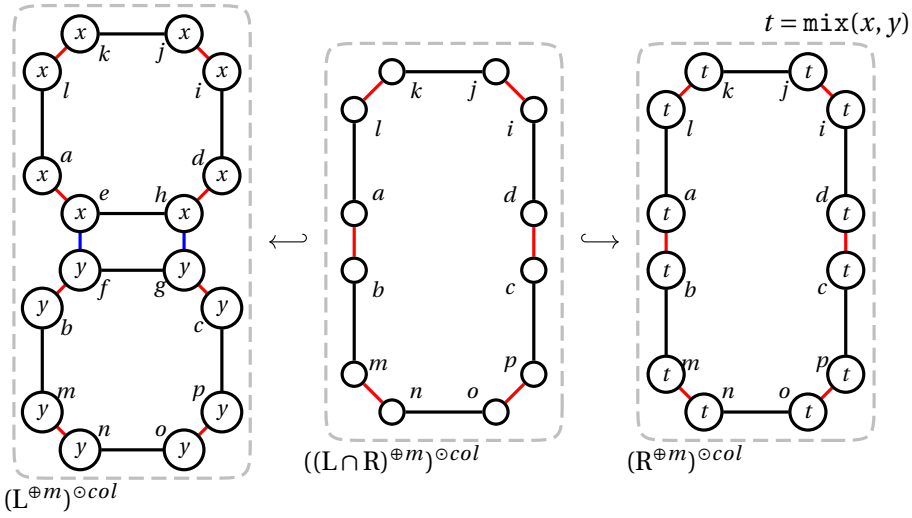


Figure 5.22: Embedding propagation of the rule from Figure 5.21.

Definition 79 (Embedding propagation of a graph)**.** *Let* $\mathrm{G} = (\mathrm{V}, \mathrm{E}, s, t, l, \Pi)$ *be a partially* Π*-embedded graph and* π *be an embedding of* Π*. The* π-embedding propagation *of* G *is the partially* Π*-embedded graph* $\mathrm{G}^{\odot\pi}$ *such for each node* v *in* V, $\pi_{\mathrm{G}^{\odot\pi}}(v) = \pi([v]_\pi)$*. The* embedding propagation *of* G *corresponds to the simultaneous* π*-embedding propagation of* G *for all* π *in* Π

The definition of partially embedded graphs ensures that each node has at most one embedding value per embedding and that all nodes in an orbit have the same embedding value (if they have one). Therefore, the embedding propagation of a partially embedded graph yields a partially embedded graph. Besides, the embeddings are represented by different attribution arcs, meaning that the embedding propagation of two distinct embeddings commutes. For π and π' in Π, and a partially embedded graph G, the propagations $(\mathrm{G}^{\odot\pi})^{\odot\pi'}$ and $(\mathrm{G}^{\odot\pi'})^{\odot\pi}$ are isomorphic graphs and simply written $\mathrm{G}^{\odot(\pi,\pi')}$.

The π-embedding propagation may not add values of type τ_π to all the nodes of $\mathrm{G}^{\odot\pi}$. Indeed, if no node of an $\langle o_\pi \rangle$-orbit is the source of an π-attribution arc in G, then no value can be added to G. For instance, the embedding propagation $\mathrm{G}^{\odot(pos,col)}$ of the graph G in Figure 5.23 is not an embedded graph. Indeed, nodes i, j, k, l, m, n, o, and p do not have position values while nodes q, r, s, and t do not have color value (depicted with white nodes). However, the embedding propagation ensures that, in an orbit, either all nodes are embedded, or no node is embedded. We call this property the all-or-nothing constraint.

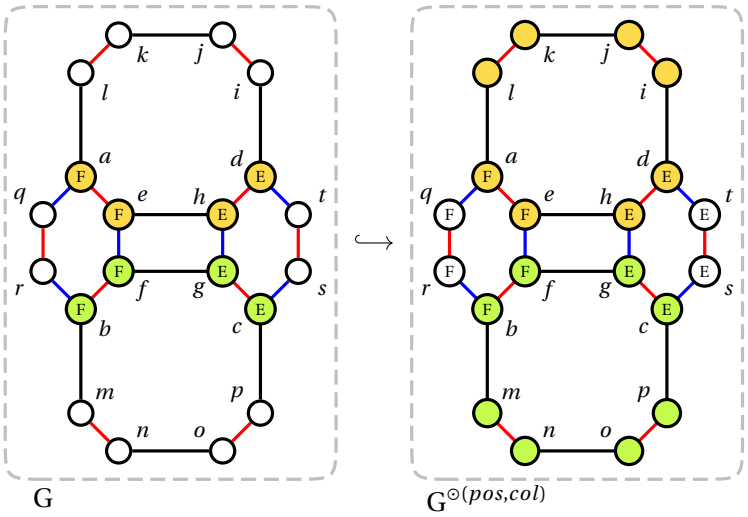


Figure 5.23: The embedding propagation only yields a partially embedded graph.

Definition 80 (All-or-nothing constraint)**.** *Let* $G = (V, E, s, t, l, \Pi)$ *be a partially* Π*-embedded graph and* π *be an embedding of* Π*. The graph* G *satisfies the* π*-all-or-nothing constraint if for all* $\langle o_\pi \rangle$*-orbits* O_π *of* G*, one of the two following properties holds:*

1. *all nodes of* O_π *share the same* τ_π*-value,*
2. *no node of* O_π *is the source of a* π*-attribution arc.*

This constraint is not to be considered as a property to be preserved by the rule but simply as a property that holds after the embedding propagation.

Proposition 5 (All-or-nothing)**.** *Let* $G = (V, E, s, t, l, \Pi)$ *be a partially* Π*-embedded graph and* π *be an embedding of* Π*. Then,* $G^{\odot\pi}$ *satisfies the* π*-all-or-nothing constraint.*

Proof. The proof is immediate from the construction of the embedding propagation. □

The graph G in Figure 5.23 correspond to the simultaneous $\langle 0, 1 \rangle$- and $\langle 1, 2 \rangle$ completion of the occurrence of the graph L from Figure 5.21b in the graph of Figure 5.21a. In other words, if we designed the edge with a translation of the extremities, we would have obtained the graph G with terms instead of values after the topological extension. Thus, a simple embedding propagation on all the graphs is not enough to obtain an embedded rule, i.e., to ensure that the left-hand and right-hand sides of the rule are embedded graphs.

We consider the all-or-nothing property as a condition on rules before saturating it with meaningless variables. We will show in Section 5.4 that the condition holds when considering the embedding propagation after the topological extension.

Definition 81 (All-or-nothing condition)**.** *The attributed rule* $r = L \xleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ *typed over* $ETG(\Pi)$ *satisfies the all-or-nothing condition for an embedding* π *in ebd f if:*

- L *and* R *satisfy the* π*-all-or-nothing constraint,*
- *for all nodes* v *in* R*, if* $\pi_R(v) = \perp$*, then* $R\langle o_\pi \rangle$ *is preserved by the rule, i.e., all its nodes and arcs are in* $L \cap R$*.*

Granted that the all-or-nothing condition holds, we can saturate a rule with meaningless variables to obtain an embedded rule.

Definition 82 (Embedding saturation)**.** *Let* $r = \mathrm{L} \xleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* ETG(Π) *and* π *an embedding of* Π*. If* r *satisfies the all-or-nothing condition for the embedding* π*, then the ebd-saturation of* r *is obtained as follows:*

1. *every* $\langle o_\pi\rangle$*-orbit of* R *without embedding values is embedded with the same fresh variable in* L, L∩R *and* R.
2. *every* $\langle o_\pi\rangle$*-orbit of* L *without embedding values is embedded with a fresh variable in* L.

If the all-or-nothing condition holds after the embedding propagation of the components of a rule r, we saturate it to obtain the complete embedding propagation of the rule.

Definition 83 (Embedding propagation of a rule)**.** *Let* $r = \mathrm{L} \xleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* ETG(Π)*. The* π*-embedding propagation of* r *is the rule* $r^{\odot\pi}$ *is a two-step process.*

Embedding propagation of the components: *The rule* $\mathrm{L}^{\odot\pi} \hookleftarrow (\mathrm{L}\cap\mathrm{R})^{\odot\pi} \hookrightarrow \mathrm{R}^{\odot\pi}$ *is obtained via the* π*-embedding in each component of* r*.*

Variable saturation *If* $\mathrm{L}^{\odot\pi} \hookleftarrow (\mathrm{L}\cap\mathrm{R})^{\odot\pi} \hookrightarrow \mathrm{R}^{\odot\pi}$ *satisfies the all-or-nothing condition, then* $r^{\odot\pi}$ *is the* π*-saturation of* $\mathrm{L}^{\odot\pi} \hookleftarrow (\mathrm{L}\cap\mathrm{R})^{\odot\pi} \hookrightarrow \mathrm{R}^{\odot\pi}$.

The embedding propagation relies on the constraints of partially embedded graphs (Definition 71), ensuring the uniqueness of the embedding value of the embedding equivalence class. If we consider an attributed rule $r = \mathrm{L} \xleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ typed over ETG(Π), where L, L∩R and R are partially embedded graphs, then an embedded match $m\colon \mathrm{L} \to \mathrm{G}$ canonically extends into an embedded match $m^{\odot\Pi}\colon \mathrm{L}^{\odot\Pi} \to \mathrm{G}$. Indeed, the constraint of orbit consistency ensures that all nodes within the same $\langle o_\pi\rangle$-orbit share the same τ_π-value in G and the same π-term in L. Therefore, for each π-attribution arcs added to $\mathrm{L}^{\odot\Pi}$ by the embedding propagation, we can find a unique π-attribution arcs in G which may serve as image. The arcs are imposed by the definition of attributed morphisms which have functions commuting with the source and target functions of an E-graph (Definition 62).

5.3.4 Complete construction

Regardless of consistency preservation, the application of typed attributed rules $r = \mathrm{L} \xleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$, where L, L∩R, and R are partially embedded graphs, to an object defined as an embedded Gmap G along a match $m\colon \mathrm{L} \to \mathrm{G}$ consists of the two steps of Figure 5.19 followed by a standard rule application. In other words, we build:

1. the topological extension $r^{\oplus m}$ along m of the rule r ;
2. the embedding propagation $(r^{\oplus m})^{\odot\pi}$ along the extended rule $r^{\oplus m}$.
3. the application of the final rule $(r^{\oplus m})^{\odot\pi}$ to the Gmap G by DPO rewriting.

Note that the embedding propagation existence depends on the satisfaction of the partial embedding constraints (Definition 71) by all extended rule parts. This existence will be ensured by conditions provided in Section 5.4.

5.4 Consistency preservation

In this section, we provide and prove conditions on attributed rules $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ typed over ETG(Π), ensuring that the topological extension and the embedding propagation yield an embedded rule. Essentially, the purpose of these two constructions was to enable the use of inconsistent rules by restoring the missing consistency conditions. In other words, the topological extension and the embedding propagation allow to weaken the conditions of embedded rules and yet obtain operations transforming embedded Gmaps into embedded Gmaps. Subsection 5.4.1 addresses the topological consistency while Subsections 5.4.2 and 5.4.3 focus on the embedding consistency. More precisely, we show that rules that satisfy some given conditions can always be extended and propagated to obtain embedded rules, i.e., rules satisfying the conditions of Definition 74.

5.4.1 Topological consistency preservation

The topological extension transforms the topological structure of the rule. Besides, topological extension is the only part of the instantiation process that modifies the rule's topological structure. We show that the topological extension transformations a rule preserves the incident arcs, non-orientation, and cycle conditions.

Theorem 9 (Topological consistency preservation of topological extension)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* ETG(Π) *and* $m\colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded Gmap.*

If r *satisfies the incident arcs, non-orientation, and cycle conditions, then the topological extension* $r^{\oplus m}$ *of* r *along the match* m *satisfies the same conditions, i.e.,*

$$r \models \mathtt{I}_r(0..n), \mathtt{O}_r(0..n), \mathtt{C}_r((0..n)_{+2}) \implies r^{\oplus m} \models \mathtt{I}_{r^{\oplus m}}(0..n), \mathtt{O}_{r^{\oplus m}}(0..n), \mathtt{C}_{r^{\oplus m}}((0..n)_{+2}).$$

In [3], the topological consistency relies on the constructions from [144]. The proof of Theorem 9 was therefore realized as a case analysis similar to the proofs of Theorems 1, 2, and 3 in Chapter 3. Here, we exploit the results of [139], presented in Chapter 3, namely the consideration of necessary and sufficient conditions.

Proof. For simplification, we develop the proof for one condition, although it holds for all three. We consider an attributed rule typed over ETG(Π) satisfying $\mathtt{I}_r(0..n)$, an embedded match $m\colon \mathrm{L} \to \mathrm{G}$ where G is an embedded Gmap, and the topological extension $r^{\oplus m}$ of r along the match m.

From Theorem 1, a rule in $\mathbb{D}$-**Graph** satisfies the weak incident arcs condition for a dimension i if and only if for all its matches $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ on a $\mathbb{D}$-graph satisfying $\mathtt{I}_\mathrm{G}(i)$, the result graph H of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ satisfies the incident arcs constraint $\mathtt{I}_\mathrm{H}(i)$.

Consider a match $m'\colon \mathrm{L}^{\oplus m} \to \mathrm{G}'$ where $\mathrm{G}' \models \mathtt{I}_{\mathrm{G}'}(0..n)$. By composition, we obtain a match

$m' \circ m_\Pi : \mathrm{L} \to \mathrm{G}'$, leading to the following diagram where all squares are pushouts.

$$\begin{array}{ccccc}
\mathrm{L} & \hookleftarrow & \mathrm{L} \cap \mathrm{R} & \hookrightarrow & \mathrm{R} \\
{\scriptstyle m_\Pi}\downarrow & & \downarrow & & \downarrow \\
(\mathrm{G}\langle o_\pi \rangle)_{\pi \in \Pi}(\mathrm{L}) & \hookleftarrow & (\mathrm{L} \cap \mathrm{R})^{\oplus m} & \hookrightarrow & \mathrm{R}^{\oplus m} \\
{\scriptstyle m'}\downarrow & & \downarrow & & \downarrow \\
\mathrm{G}' & \hookleftarrow & \mathrm{D}' & \hookrightarrow & \mathrm{H}'
\end{array}$$

From Theorem 1, the result graph H' of the direct derivation $\mathrm{G}' \Rightarrow^{r, m' \circ m_\Pi} \mathrm{H}'$ satisfies the incident arcs constraint $\mathtt{I}_{\mathrm{H}'}(0..n)$. The graph H' is (isomorphic to) the result graph of the direct derivation $\mathrm{G}' \Rightarrow^{r^{\oplus m}, m'} \mathrm{H}$. Since this result holds for all graphs H', the reverse sense of Theorem 1 ensures that the topological extension $r^{\oplus m}$ of r along the match m satisfies $\mathtt{I}_{r^{\oplus m}}(0..n)$.

The proof holds for all conditions, meaning that the topological extension preserves the topological conditions from Chapter 3. □

As the embedding propagation only modifies attribution arcs, it does not alter the topology, thus preserving the topological consistency.

Theorem 10 (Topological consistency preservation of the embedding propagation)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* $\mathrm{ETG}(\Pi)$*, where* L, $\mathrm{L} \cap \mathrm{R}$ *and* R *are partially embedded graphs.*

If r *satisfies the incident arcs, non-orientation, and cycle conditions, then the* Π*-embedding propagation* $r^{\odot \Pi}$ *of* r *satisfies the same conditions, i.e.,*

$$r \models \mathtt{I}_r(0..n), \mathtt{O}_r(0..n), \mathtt{C}_r((0..n)_{+2}) \implies r^{\odot \Pi} \models \mathtt{I}_{r^{\odot \Pi}}(0..n), \mathtt{O}_{r^{\odot \Pi}}(0..n), \mathtt{C}_{r^{\odot \Pi}}((0..n)_{+2}).$$

Proof. The embedding propagation step only modifies nodes and arcs without change to the attribution arcs and the data nodes. □

From Theorems 9 and 10, the preservation can be extended to the whole process defined in Section 5.3. If a type attributed rule r, consisting of partially embedded graphs, satisfies the incident arcs, non-orientation, and cycle conditions, then the Π-embedding propagation of the topological extension $(r^{\oplus m})^{\odot \Pi}$ satisfies the same conditions.

5.4.2 Condition of non-overlap

Before we study the preservation of the embedding consistency, we mention a particularly undesirable situation likely to occur: the overlap of embedding orbits. By considering a minimal match of the transformed embeddings that relies on the automatic completion of transformed embedding orbits, we are exposed to unexpected merges of different embedding orbits. Let us consider the face stretching defined by the rule of Figure 5.24.

The operation matches two edges, defined by the orbits $\langle 0 \rangle(a)$ and $\langle 0 \rangle(d)$, to translate vertices at their extremities. The pos-embedding terms $\overrightarrow{v}$ and $-\overrightarrow{v}$ translate the edge in opposite directions. In Figure 5.25a, these orbits are matched to opposite edges of a square face (the match is specified by the identifiers and highlighted with colors). The extended rule of Figure 5.25e contains four vertices in R respectively embedded by four distinct terms. The match from L to the

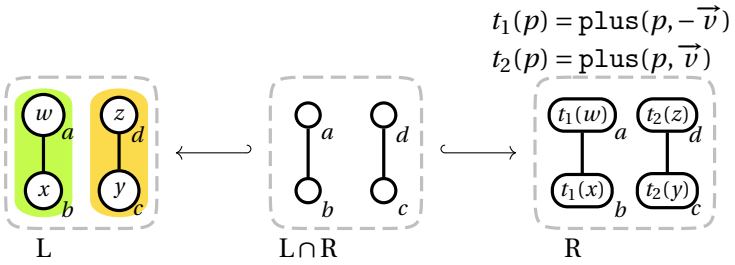


Figure 5.24: Face stretching rule.

graph of Figure 5.25a maps the variables as follows: $w \mapsto$ B, $x \mapsto$ C, $y \mapsto$ D, and $z \mapsto e$. Therefore, we obtain the updated positions $\text{B}' = \text{B} - \vec{v}$, $\text{C}' = \text{C} - \vec{v}$, $\text{D}' = \text{D}\,\vec{v}$ and $\text{E}' = \text{E} + \vec{v}$, as displayed in Figure 5.25b.

The operation matches two edges, defined by the orbits $\langle 0 \rangle(a)$ and $\langle 0 \rangle(d)$, to translate vertices at their extremities. The pos-embedding terms $\vec{v}$ and $-\vec{v}$ translate the edge in opposite directions. In Figure 5.25a, these orbits are matched to opposite edges of a square face (the match is specified by the identifiers and highlighted with colors). The extended rule of Figure 5.25e contains four vertices in R respectively embedded by four distinct terms. The match from L to the graph of Figure 5.25a maps the variables as follows: $w \mapsto$ B, $x \mapsto$ C, $y \mapsto$ D, and $z \mapsto e$. Therefore, we obtain the updated positions $\text{B}' = \text{B} - \vec{v}$, $\text{C}' = \text{C} - \vec{v}$, $\text{D}' = \text{D}\,\vec{v}$ and $\text{E}' = \text{E} + \vec{v}$, as displayed in Figure 5.25b.

However, when the match maps nodes a and d to nodes within the same $\langle 0, 1 \rangle$-orbit, the extended rule is inconsistent since a vertex gets two separate terms for the position embedding. For instance, the application of the rule of Figure 5.24 is inconsistent along the match of Figure 5.25d, defined by identifiers and highlighted in with colors. The extended rule of Figure 5.25f is inconsistent as the top vertex ends up embedded in R with two different terms. If we apply the rule, disregarding the consistency, we obtain two updated positions $\text{A}' = \text{A} - \vec{v}$ and $\text{A}'' = \text{A} + \vec{v}$ for the top vertex, as illustrated in the intuitive description of Figure 5.25c. Such an example is a clear case of misapplication as we wanted to match and translate four vertices but only match three. We call an overlap such a situation where different embedding orbits manipulated in the rule end up merged in the extended rule.

Definition 84 (Overlap in a morphism)**.** *Let* $m\colon \text{H} \to \text{G}$ *an embedded morphism typed over* ETG(Π) *where* G *is a partially embedded combinatorial graph and* H *is a partially embedded graph. The morphism* m *has an* overlap *for an embedding* π *in* Π *if there exist two nodes* v *and* w *of* H*, such that* $v \not\equiv_\pi w$ *in* H *and* $\pi_\text{H}(v) \neq \pi_\text{H}(w)$*, but* $m(v) \equiv_\pi m(w)$ *in* G*.*

This construction directly extends to rules by considering the match and comatch.

Definition 85 (Rule producing an overlap)**.** *Let* $r = \text{L} \xleftarrow{i_\text{L}} \text{L} \cap \text{R} \xhookrightarrow{i_\text{R}} \text{R}$ *be an attributed rule typed over* ETG(Π)*, where* L, L ∩ R *and* R *are partially embedded graphs, and* $m\colon \text{L} \to \text{G}$ *an embedded match where* G *is an embedded combinatorial graph. Let* H *be the result of the direct derivation* $\text{G} \Rightarrow^{r,m} \text{H}$ *if it exists, and* $m'\colon \text{R} \to \text{H}$ *the comatch of the double pushout diagram. The applications of the rule* r *along* m produces an overlap *for an embedding* π *in* Π *if* m *or* m' *has an overlap for* π*.*

We strive to obtain rules without overlap. Note that overlaps should be checked dynamically and not statically. Indeed, a rule may produce an overlap when applied to some graphs but not

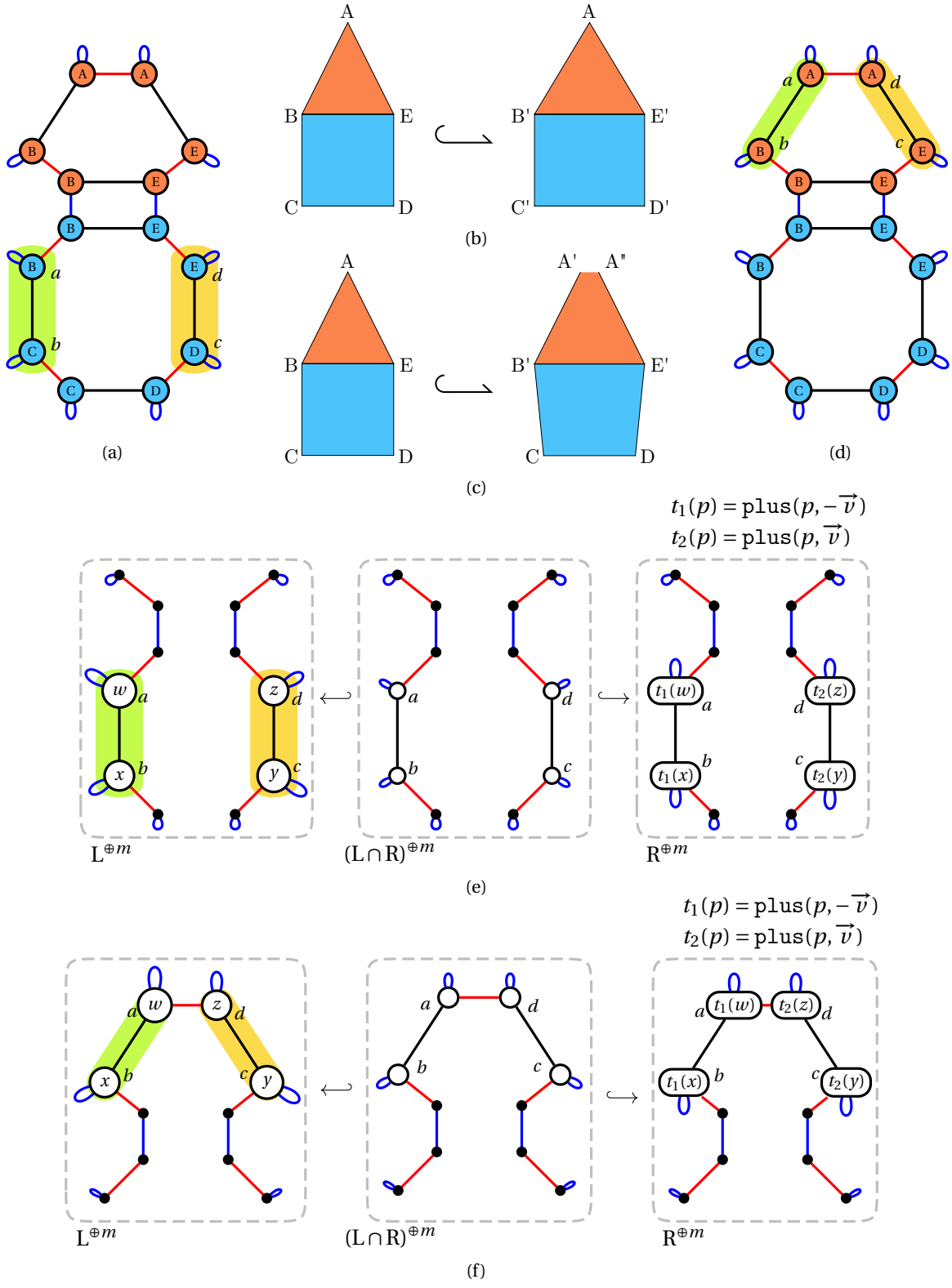


Figure 5.25: Overlaps in the operation of face stretching: (a) a match that does not produce an overlap, (b) consistent face stretching, (c) inconsistent face stretching, (d) a match that produces an overlap, (e) topological extension for the non-overlapping match, and (f) topological extension for the overlapping match.

others, e.g., the rule performing the face stretching in Figure 5.25. We define a condition on the match to ensure that the topological extension does not produce any overlap. This condition can be seen as an extension of the injectivity condition of the embedded matches (Definition 73) to the embedding orbits. Intuitively, the condition prevents connecting two non-equivalent nodes

by elements outside the rule. In this case, nodes are considered equivalent for an embedding if they either belong to the same embedding orbit or share the same embedding term.

Lemma 12 (Non-overlap)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed combinatorial rule typed over* $\mathrm{ETG}(\Pi)$*, where* L, $\mathrm{L}\cap\mathrm{R}$ *and* R *are partially embedded graphs and* $m\colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded combinatorial graph.*

The applications of the rule r *along* m *does not produce any overlap if for all embedding* π *in* Π*, for any two nodes* v *and* w *of* $\mathrm{L}\cap\mathrm{R}$ *such that either* $v \not\equiv_\pi w$ *in* R *and* $\pi_\mathrm{R}(v) \neq \pi_\mathrm{R}(w)$*, or* $v \not\equiv_\pi w$ *in* L *and* $\pi_\mathrm{L}(v) \neq \pi_\mathrm{L}(w)$*, then* $m(v) \not\equiv_\pi m(w)$ *in* G *where the equivalence is considered only with arcs in* $\mathrm{E_G} \setminus m_\mathrm{E}(\mathrm{E_L})$*.*

Lemma 12 states that a weaker property can ensure the absence of overlap. First, it suffices to verify the absence of overlap for the nodes in $\mathrm{L}\cap\mathrm{R}$ rather than in L and R. Secondly, we can check all overlaps directly in G rather than in both G and H by only considering the arcs outside the occurrence of the match. Considering the equivalence with arcs outside of the match occurrence means nodes could be equivalent in G, but the equivalence is broken by the rule, such as in Figure 5.26. For instance, the match $m\colon \mathrm{L} \to \mathrm{G}$ maps the $\langle 1\rangle$-orbits of L into distinct $\langle 0, 1\rangle$-orbits in G in Figure 5.26. However, the rule modifies the topology of the object, and all nodes belong to the same $\langle 0, 1\rangle$-orbit in $\mathrm{R}^{\oplus m}$. Hence, the equivalence is to be considered outside the occurrence of the match, i.e., with only the arcs not highlighted in G.

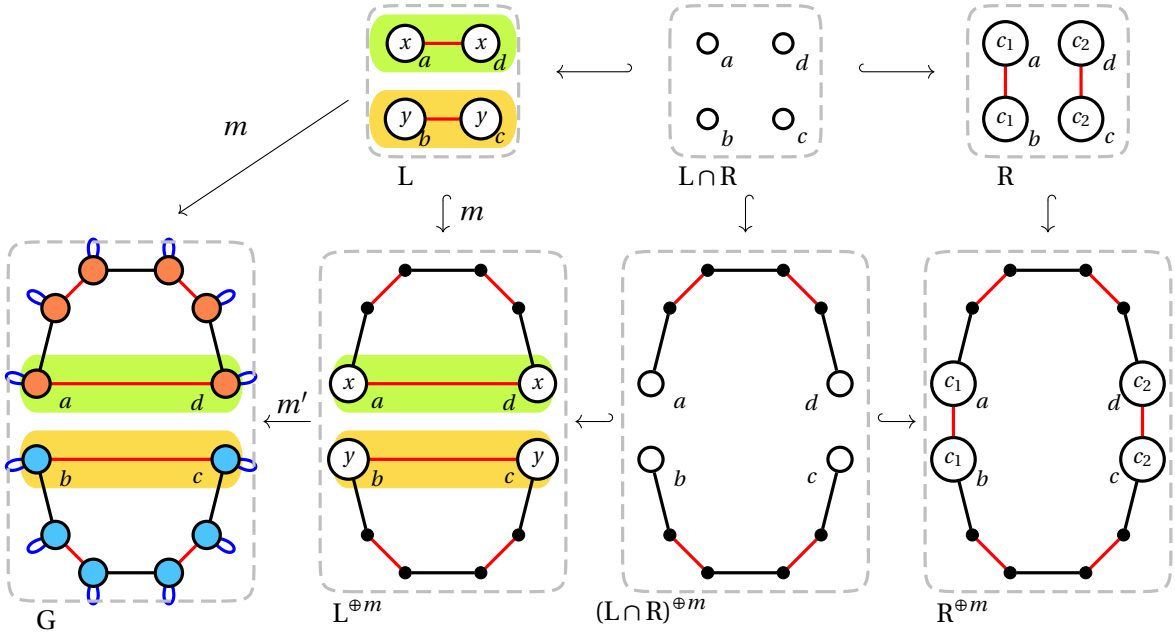


Figure 5.26: Modifications of the topology may produce overlaps.

Proof. Let $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L}\cap\mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ be an attributed rule typed over $\mathrm{ETG}(\Pi)$, where L, $\mathrm{L}\cap\mathrm{R}$ and R are partially embedded graphs, $m\colon \mathrm{L} \to \mathrm{G}$ an embedded match where G is an embedded combinatorial graph, and π an embedding in Π. We write m' for the comatch $\mathrm{R} \to \mathrm{H}$.

For the left-hand side, the proof is pretty straightforward and results from the fact that the incident arcs property ensures that a deleted node belongs to the same orbit as a preserved node unless it belongs to a full orbit. Formally, if two v and w of L, satisfy $v \not\equiv_\pi w$ in L, $\pi_\mathrm{L}(v) \neq \pi_\mathrm{L}(w)$ and $m(v) \equiv_\pi m(w)$ in G. Then, the incident arcs condition of the combinatorial rules ensures that

there exist nodes v' and w' in L∩R such that $v' \equiv_\pi v$ and $w' \equiv_\pi w$ in L. By hypothesis on embedded rules, it holds that $\pi_L(v') \neq \pi_L(w')$ and $m(v') \equiv_\pi m(w')$ in G, which contradicts the hypothesis of the lemma. Thereafter, we obtain the part of the lemma on L.

For the right-hand side, the proof is harder since we need to change the morphism, i.e., analyze content in G rather than in H. Let v and w be two nodes of R such that $v \not\equiv_\pi w$ in R and $\pi_R(v) \neq \pi_R(w)$.

▶ *Modified orbit*

If two nodes v' and w' exists in L∩R such that $v' \equiv_\pi v$ and $w' \equiv_\pi w$. Since v' is in the same $\langle o_\pi \rangle$-orbit as v in R, resp. w' is in the same $\langle o_\pi \rangle$-orbit as w, but v and w have disjoint orbits, it holds that $v' \not\equiv_\pi w'$ in R. Since $\pi_R(v) \neq \pi_R(w)$, the construction of embedded rules guarantees that $\pi_R(v') \neq \pi_R(w')$. Therefore, we can apply the hypothesis of the lemma, meaning that $m(v') \not\equiv_\pi m(w')$ in G where the equivalence is considered only with arcs in $E_G \setminus m_E(E_L)$. The arcs of $E_G \setminus m_E(E_L)$ are not modified by the rule, and we obtain the counterpart property in H: $m'(v') \not\equiv_\pi m'(w')$ in H where the equivalence is considered only with arcs in $E_H \setminus m'_E(E_R)$. Besides, m' is injective on the topological content of the graphs. Therefore, the non equivalence $v' \not\equiv_\pi w'$ in R induces a non equivalence $m'(v') \not\equiv_\pi m'(w')$ in H. The reasoning in R holds in H by injectivity of m' on the topological content. In particular, $m(v)$ is in the same $\langle o_\pi \rangle$-orbit as $m(v')$ in H, resp. $m'(w)$ is in the same $\langle o_\pi \rangle$-orbit as $m'(w')$, but $m'(v')$ and $m'(w')$ have disjoint orbits in H, meaning that $m'(v) \not\equiv_\pi m'(w)$ in H.

▶ *Added orbit*

Otherwise, either v or w belong to an orbit completely added, i.e., with all nodes in R \ L. Without loss of generality, we assume it is v. Then, all nodes in $H\langle o_\pi \rangle(m'(v))$ are nodes of $m'(R)$. In particular, for all nodes y in H such that $y \equiv_\pi m'(v)$ in H, there is a unique node x in R such that $m'(x) = y$. For these nodes x, it holds that $x \equiv_\pi v$ in R. Since $w \not\equiv_\pi v$ in R, $m'(w) \notin H\langle o_\pi \rangle(m'(v))$, i.e., $m'(v) \not\equiv_\pi m'(w)$ in H.

Thereafter, the application of the rule r along m does not produce any overlap. □

The following lemma ensures that we can check the existence of overlaps in the extended rule rather than on the complete graph.

Lemma 13 (Overlap can only exist in the topological extension)**.** *The applications of a rule* $r = L \xhookleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ *along* $m : L \to G$ *in the conditions of Definition 85 produces an overlap for an embedding* π *if and only if it produces the applications of* r *along* $m_\Pi : L \to L^{\oplus m} = (G\langle o_\pi \rangle)_{\pi \in \Pi}(L)$ *(Definition 77) produces the same overlap.*

The lemma holds because the topological extension completes the orbit.

Proof. The topological extension completes the orbits of L, meaning that the considered nodes are identical. For the right-hand side, the construction of the topological extension ensures that for two nodes v and w in R, $m'(v) \equiv_\pi m'(w)$ in H is equivalent to $m'_\Pi(v) \equiv_\pi m'_\Pi(w)$ in $R^{\oplus m}$, where $m'_\Pi : R \to R^{\oplus m}$ is the comatch of the double pushout diagram constructing $r^{\oplus m}$. □

Therefore, we can restrict the study of overlaps to the topological extension of the rule and say that the topological extension produces an overlap if the application of the rule along the completed embedded match morphism (Definition 77) produces an overlap. The condition of non-overlap will be required to prevent unwanted applications before considering the embedding consistency preservation.

5.4.3 Embedding consistency preservation

We now study how the non-overlap condition combined with the topological extension and embedding propagation allows for weaker conditions on rules while still ensuring the transformation of an embedded Gmap into an embedded Gmap. The starting point is a rule satisfying a weakly embedded combinatorial rule (Definition 74), i.e., a rule with at most one π-value per $\langle o_\pi \rangle$-orbit. The topological extension restores the condition of the full orbits of transformed embeddings, meaning that orbits with modified embedding values are full orbits. However, the topological extension restores this condition at the cost of obtaining partially embedded components. In other words, we obtain a partially embedded combinatorial rule (Definition 74). Finally, the embedding propagation restores the complete conditions of embedded rules. Therefore, we obtain a solution to remove the condition of the full orbits of transformed embeddings which was the goal of the constructions introduced in Section 5.3.

We start with the topological extension. We consider combinatorial rules to allow for considering arcs from their dimension but remove the condition of the full orbits of transformed embeddings, leading to weakly embedded combinatorial rules (Definition 74). The extension adds nodes without embedding values, i.e., nodes that are missing some π-attribution arcs for some embedding π in Π. Therefore, the topological extension only provides partially embedded (combinatorial) rules.

Theorem 11 (Embedding consistency preservation of topological extension)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be a weakly* Π*-embedded combinatorial rule and* $m \colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded combinatorial graph.*

If the topological extension $r^{\oplus m}$ *of* r *along the match* m *produces no overlap for any embedding* π *of* π*, then* $r^{\oplus m}$ *is a partially* Π*-embedded combinatorial rule.*

A partially embedded rule only requires partially embedded components, meaning each node is the source of a unique π-attribution arc. Indeed, these arcs will later be added by the embedding propagation. However, the condition of the full orbits of transformed embeddings should be restored.

Proof. We provide a sketch of the proof, while a complete version is presented in Appendix B.2. The condition of partially embedded components holds since the topological extension adds nodes without embedding or overlap to an initial weakly embedded rule that satisfies the condition of embedded components. The condition of the full orbits of transformed embeddings directly results from the topological extension that adds all necessary nodes to complete the partial orbits. □

Let us now show that the embedding propagation (Definition 79) of the topological extension of a rule is well-defined. Recall that the embedding propagation of a rule is a two-step process.

First, we perform the embedding propagation of the rule's left-hand side, interface, and right-hand side. Secondly, assuming that this rule satisfies the all-or-nothing constraint (Definition 81), we add variables to obtain an embedded rule.

Lemma 14. *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be a weakly* Π*-embedded combinatorial rule,* $m\colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded combinatorial graph,* $r^{\oplus m}$ *the topological extension of* r *along the match* m*, and* π *an embedding of* $ebdf$*. Then, the* π*-embedding propagation* $(r^{\oplus m})^{\odot \pi}$ *of* $r^{\oplus m}$ *is well-defined.*

Proof. The π-embedding propagation is a two-step process. The embedding propagation of the components is subject to no hypothesis and is, therefore, well-defined. After the embedding propagation, the embedding saturation (Definition 82) of the rule requires that the all-or-nothing condition holds (Definition 81). Thus, we show that the embedding propagation of the components of $r^{\oplus m}$ satisfies the all-or-nothing condition (Definition 81).

From Property 5, the embedding propagation on the components of $r^{\oplus m}$ yields graphs $(\mathrm{L}^{\oplus m})^{\odot \pi}$ and $(\mathrm{R}^{\oplus m})^{\odot \pi}$ satisfying the all-or-nothing constraint (Definition 80). All their $\langle o_\pi \rangle$-orbits are either fully embedded or not embedded at all. Besides, if a node v of $(\mathrm{R}^{\oplus m})^{\odot \pi}$ is the source of no π-attribution arc, then so are all the node in the orbit $(\mathrm{R}^{\oplus m})^{\odot \pi}\langle o_\pi \rangle(v)$ (otherwise the embedding propagation would have added attribution arcs). In particular, these nodes were all added by the topological extension and therefore are preserved in the span. Therefore, the embedding propagation of the components of $r^{\oplus m}$ satisfy the all-or-nothing condition (Definition 81).

The embedding propagation of the topological extension is well-defined. □

Let us now show that the embedding propagation (Definition 79) restores the original embedding consistency conditions, in particular, the condition of the full match of the transformed embeddings.

Theorem 12 (Embedding consistency preservation of the embedding propagation)**.** *If* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *is an attributed combinatorial rule typed over* ETG(Π) *satisfying the conditions of partially embedded components and full orbits of transformed embeddings, then* Π*-embedding propagation* $r^{\odot \Pi}$ *of* r *is a* Π*-embedded combinatorial rule.*

Proof. This proof is rather straightforward. The embedding propagation of components adds terms based on existing terms in orbits, while the embedding saturation adds the same term to all nodes within an orbit. Thus, the embedding propagation preserves the constraint of orbit consistency on its components. Furthermore, the embedding saturation adds all missing attribution arcs to nodes of the left-hand and right-hand sides. Therefore, the embedding propagation of the topological extension satisfies the condition of embedded components. Similarly, the embedding propagation preserves the condition of the full orbits of transformed embeddings by reasoning on the equivalence class of the embedding equivalence relation. □

From the results of the theorems and lemma presented in this section, we can derive rules preserving the topological and embedding consistency of embedded Gmaps from weaker properties than in Theorem 8.

Theorem 13 (Preservation of the embedding consistency for weakly embedded combinatorial rule)**.** *Let* $r = \mathrm{L} \xhookleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ *be an attributed rule typed over* $\mathrm{ETG}(\Pi)$ *and* $m\colon \mathrm{L} \to \mathrm{G}$ *an embedded match where* G *is an embedded Gmap.*

If r satisfies:

- *the incident arcs condition* $\mathtt{I}_r(0..n)$ *(Definition 34),*
- *the non-orientation condition* $\mathtt{O}_r(0..n)$ *(Definition 37),*
- *the cycle condition* $\mathtt{C}_r((0..n)_{+2})$ *(Definition 40),*
- *the condition of embedded components (Definition 74),*

then $(r^{\oplus m})^{\odot\Pi}$*, the* Π*-embedding propagation of the topological extension of r along the match m, transforms an embedded Gmap into an embedded Gmap.*

Proof. The proof holds from Theorems 9 10 for the topological part and from Theorems 11 and 12 for the embedding part. □

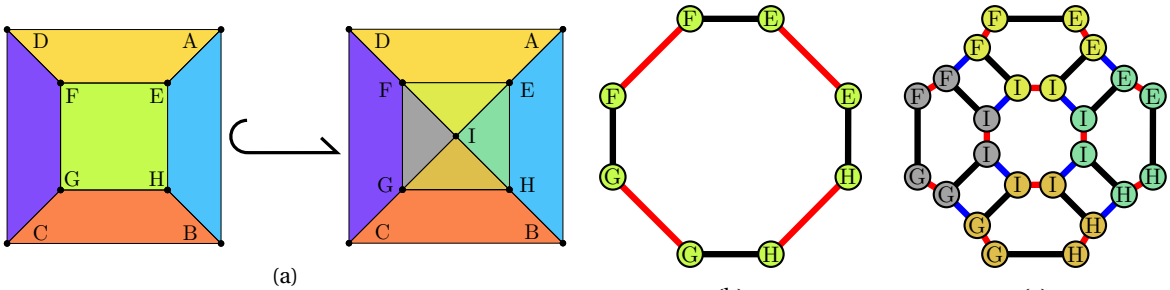
This proof concludes our solution to the first restriction discussed at the end of Section 5.2. We modified the application of a rule via the topological extension and the embedding propagation such that embedding values may be modified even though the complete corresponding orbit was not present in the rule. Note that the modifications of the rewriting step come with a downside: the rule can no longer be checked statically at design time. Indeed, the condition of non-overlap has to be checked dynamically, depending on the match. We will now deal with the second restriction, i.e., the access of values through the topological structure.z

5.5 Accessing values through the topological structure

The embedded rules defined so far allow a consistent rewriting of embedded Gmap. In Sections 5.3 and 5.4, we proposed a mechanism to dynamically extend rules based on the object under modification. This automatic completion enables more compact rules that encode possibly infinitely many embedded rules. However, even with this mechanism, the obtained rules still do not correspond to standard operations in geometric modeling. Indeed, geometric computations may require access to values outside the rule's scope. These accesses are typically described in topological terms, e.g., the color of the adjacent face or the positions of the vertices opposite of the incident edges. However, as for the orbit completion, we might need an undetermined number of embedding values for computation, for instance, all the positions in a face. In the case of positions, it is usually a fair assumption to consider them all distinct. However, the assumption does not hold when considering other embeddings, such as colors, where the same embedding value can appear multiple times. In other words, operations modifying the geometry of an object may require accessing collections of embedding values and considering the multiplicities of the values. We solve these issues by modification of the data signature. We first deal with graph traversals, then multisets, and finally collect operators exploiting traversals for building multisets.

5.5.1 Topological traversals

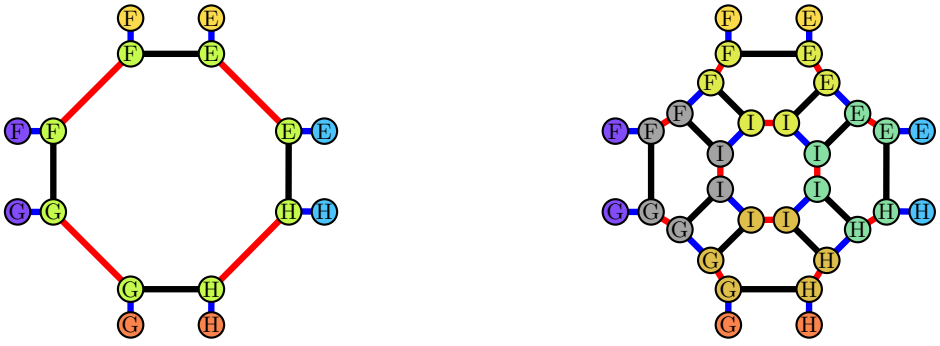
Computing new embedding values requires access to the existing ones through the topological structure. For instance, we may want to gather all positions of the vertices of a face without having the entire face in the rule. Before presenting constructions for topological traversals, we illustrate the difficulties they generate. We consider the barycentric triangulation presented at the beginning of the chapter and illustrated in Figure 5.27. An intuitive description is provided in Figure 5.27a, while the modified parts are given in Figures 5.27b and 5.27c.



(a) (b) (c)

Figure 5.27: Barycentric triangulation of an inner face with color modification: (a) intuition, (b) modified part of the Gmap before the operation, and (c) after the operation,

If we were to consider a rule from the topological structure of the part before modification from Frigure 5.27b and 5.27c, we would not be able to express the geometric modification. Indeed, the new face colors are computed as the mix of the initial color and the face of the adjacent face. We can access the adjacent faces from the initial nodes by traversing the 2-arcs. Thus, we would need to extend the occurrence of the match to the graph of Figure 5.28a, obtaining the graph of Figure 5.28b as the occurrence of the comatch.



(a) (b)

Figure 5.28: An extension of the modified parts allowing access to the adjacent faces' color: (a) modified part of the Gmap before the operation, and (b) after the operation.

Although this extension may appear anecdotal, modeling operations may require accessing geometric values far from the modified part. For instance, the butterfly subdivision scheme [51] access four arcs away from the initial face for computing the positions of the new vertices. In other words, we may need to extend the rule with arbitrarily many elements to be able to access the embedding values. The issue is aggravated by possible self-overlap in these elements. For instance, an edge on the boundary of the objects consists of 2 loops, meaning that the extension of Figure 5.28 would not be applicable. Besides, considering multisets of values would result in

infinitely many extensions, i.e., we get back the issue solved by the topological extension. However, the issue here only comes from the necessity to access embedding values, not update them.

To our knowledge, graph traversals in graph databases [155] are the only line of research exploiting traversals of the graph structure for attribute computation. In this approach, the traversal is written as a chain of queries on the graph. However, these traversals are used for database queries, not database editing. Note that simultaneous edits and queries are not intended in database management. Besides, traversals described in [155] would suffer from the same issues pointed out previously: the edition of the graph structure through a rule may modify the traversal. In [3], we extended the algebra with a type `Node` whose carrier set matched the set of nodes in the initial Gmap. Our reformulated framework prohibits this possibility because the algebra should be preserved through a transformation and because the set of nodes is assumed disjoint from the carrier sets of the algebra.

To summarize, we are facing two issues. First, we need to access values outside the occurrence of the match. Requiring the designer to extend the rule with each possible topological extension would remove the benefits gained from the topological extension and embedding propagation. Secondly, these accessed values should always be accessed in the initial graph. Thus, (algebraic) functions to traverse the structure do not offer a valid solution since they would result in different computations for the match and the comatch.

Without a proper algebraic solution to our problem, we propose to construct a special kind of variables to be evaluated in a specific way by the match. This solution allows traversing the initial graph even if the terms appear in the right-hand side of the rule. We propose to use an identifier attribute that uniquely defines nodes similar to the solution mentioned in [98, Chap. 1] to handle user IDs or registration numbers in a fashion similar to database keys. However, these identifiers are to be replaced by variables used as terms in the rule. We start with variables for topological traversals and will deal with multisets afterward. For a set of embedding Π, we consider the signature $\Omega_\Pi(\mathtt{id}) = (S_\Pi(\mathtt{id}), F_\Pi(\mathtt{id}))$ such that:

- $S_\Pi(\mathtt{id}) = \{\tau_\pi \mid \pi \in \Pi\} \cup \{\mathtt{id}\}$,
- $F_\Pi(\mathtt{id}) = \{@\mathtt{i} \mid i \in 0..n\} \cup \{\pi \mid \pi \in \Pi\}$.

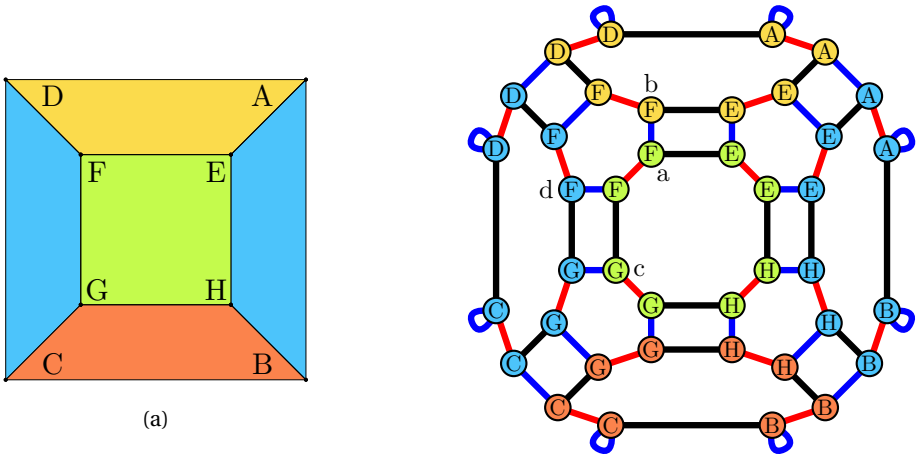
The new type `id` is intended to store the node's ID and allow accessing nodes of the modified graph. The function names `@i` have a profile $\rho(@\mathtt{i}) = \mathtt{id} \to \mathtt{id}$, with i a dimension in $0..n$, and encode navigation through the topological structure. Similarly, the function names π describe the access of embedding values and have a profile $\rho(\pi) = \mathtt{id} \to \tau_\pi$, where π is an embbeding of Π.

We now build $\Omega_\Pi(\mathtt{id})$-algebras. Such an algebra depends on an underlying Gmap and exploits its regularity. This regularity allows for characterizing paths based on words of dimensions (as already seen in Chapter 4) and defining an $\Omega_\Pi(\mathtt{id})$-algebra $\mathscr{A}(G)$ based on a Π-embedded Gmap G. The main idea is then to replace the algebra of a Π-embedded Gmap G with $\mathscr{A}(G)$. Thus, let us consider a Π-embedded Gmap $G = (\mathscr{G}_G, \mathscr{A}_G)$ where $\mathscr{G}_G$ is an E-graph and $\mathscr{A}_G$ is a $\Omega(\Pi)$-algebra. First, we extend the embedding type graph ETG(Π) with an attribution arc id with target a data node `id` and add a id-attribution arc to each node of $\mathscr{G}_G$. We assume their target to be distinct. In practice, Sid can be viewed as an embedding $\langle\rangle \to \mathtt{id}$. Secondly, we build the a Secondly, we build the algebra $\mathscr{A}(G)$. For the carrier sets, we consider the same sets as $\mathscr{A}_G$ with the addition of $\mathscr{A}(G)_{\mathtt{id}} = \mathtt{UUID}$, a set of identifier values. Although we simply extended the type names from

$\Omega(\Pi)$ to $\Omega_\Pi(\texttt{id})$, the set of function names has been completely replaced. In $\mathscr{A}(G)$, we consider functions $@i^{\mathscr{A}(\Pi)}$, for each dimension i. The function $@i^{\mathscr{A}(\Pi)}$ gives access to the identifier of the unique i-neighbor of a node given by its identifier. Formally, for a dimension i in $0..n$, and a node v in V, $@i^{\mathscr{A}(\Pi)}(id(v)) = id(t(e))$, such that e is the unique i-arc $e \in E$ satisfying $s(e) = v$. We also consider functions $\pi^{\mathscr{A}(\Pi)}$ for each embedding π in Π. The function yields the embedding value of a node given by its identifier, i.e., $\pi^{\mathscr{A}(\Pi)}(id(v)) = \pi(v)$ for a node v, where $\pi(v)$ is the target of the unique π-attribution arc of source v. The algebra $\mathscr{A}(G)$ provides a solution to compute traversals in G. We use the term algebra $T_{\Omega_\Pi(\texttt{id})}(X)$ to construct expressions describing traversals and access to embedding values.

Example 76 (Terms in $T_{\Omega_\Pi(\texttt{id})}(X)$)**.** Let X be a set of id variables containing a. Then, the terms $pos(a)$, $col(a)$, $col(@2(a))$, $pos(@0(@1(a)))$, and $col(@2(@1(a)))$ are valid terms of $T_{\Omega_\Pi(\texttt{id})}(X)$.

We consider the evaluation of these terms on the algebra $\mathscr{A}(G)$ where G is the Gmap of Figure 5.29b, under the assumption that the variable a of type id is mapped to the id of the node a. The terms $pos(a)$ and $col(a)$ respectively provide access to the position and color of the node a and evaluate as $pos(a) = F$ and $col(a) =$ ●. The term $col(@2(a))$ access a color embedded after traversing a 2-arc. From node a, the traversal ends in node b. Thus, the term evaluate as $col(@2(a)) =$ ●. Likewise, the term $pos(@0(@1(a)))$, resp. $col(@2(@1(a)))$, access a position, resp. a color, value after traversing a 10, resp. 12, path. They evaluate as $pos(@0(@1(a))) = G$ and $col(@2(@1(a))) =$ ●.



(a) (b)

Figure 5.29: An object with two faces sharing the same color: (a) the object, and (b) the embedded Gmap.

Any term t from of $T_{\Omega_\Pi(\texttt{id})}(X)_{\tau_\pi}$ for an embedding π in Π is called a π-*identifier term*. Recall from Definition 59, that a term from $T_{\Omega_\Pi(\texttt{id})}(X)_{\tau_\pi}$ is a term of type τ_π with variables in X. By construction of $\Omega_\Pi(\texttt{id})$, any variable in X has type id and any π-identifier term describes the access to the π-embedding value of a node through a traversal. We write $T_\Pi(X) = \cup_{\pi\in\Pi} T_{\Omega_\Pi(\texttt{id})}(X)_{\tau_\pi}$ for the set of all identifier terms. These identifier terms can be evaluated within a $\mathscr{A}(G)$, provided an embedding Gmap G. The evaluation can be canonically built from mapping the variables X.

We now use identifier terms as variables in an Π-embedded rule. Since those variables have types in $S(\Pi)$, the construction is valid. In particular, the sets of function names are disjoint, meaning we can unambiguously read a term. For instance, the term $t = \texttt{plus}(pos(id_a), \vec{v})$ has variables $Var(t) = \{pos(id_a), \vec{v}\}$. The variable $pos(id_a)$ has type pos. It also constitutes a π-identifier term.

Since $\overrightarrow{v}$ is not an identifier term, t is a term of type pos with variables in $T_\Pi(X) \sqcup Y$. In other words, we consider terms that contain both identifier terms 'regular' variables.

The evaluation of rules with terms using identifier terms as variables requires providing meaning to the identifier term, i.e., mapping their variables of type `id`. Thus, we need to extend the embedding of a Gmap with `id` attributes. Similar to the construction of $\Omega_\Pi(\texttt{id})$-embedded Gmaps, we add id-attribution arcs to the Gmap (and the embedding type graph), viewed as an embedding $\langle\rangle \rightarrow \texttt{id}$. However, we only extend the set of type names and not the set of function names (such that terms are still unambiguous). An $\Omega(\Pi)$-algebra thus comes with an additional carrier set `UUID` of type `id` but no additional functions. We can now rewrite the vertex translation rule of Figure 5.14 as in Figure 5.30a, where all embbedings are specified in the nodes with expressions of the form $\pi = t$ where π is an embedding and t a term of type τ_π.

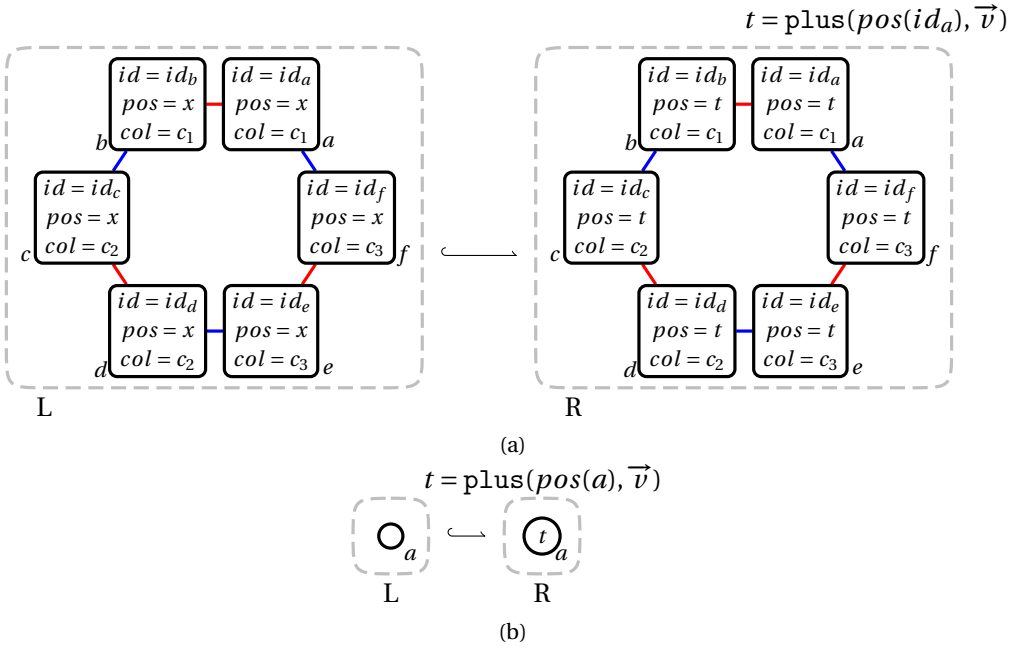


Figure 5.30: Vertex translation rule with the type id: (a) the complete rule, (b) a compact representation.

Note that all non-modified embeddings are redundant. Here, the id and col embeddings need not be specified. Furthermore, using identifiers means that variables for embeddings different than id are unnecessary. We could indeed replace c_1 by $col(id_a)$, c_2 by $col(id_c)$, and c_1 by $col(id_e)$ in R. Account for the topological extension and the embedding propagation, we obtain the more compact representation of Figure 5.30b. In this representation, we also merge the notion of identifier embedding id and the node identifier describing the rule morphisms. Since we impose that the left-hand and right-hand sides of rules are embedded graphs, this representation offers an easier-to-visualize rule. Formally, nodes always have embedding values for each embedding, i.e., the graph L in Figure 5.30b is considered embedded, by opposition to the rule from Figure 5.15b.

A more consequent example will be described once we solve the issue of accessing multiple values within the Gmap. However, we first need to define multisets as an algebraic construction.

5.5.2 Data types with multisets

We propose a minimal extension of the data signature to be able to deal with collections storing multiple occurrences of embedding values. For each data type τ, we construct the type $\mathcal{M}(\tau)$ that

corresponds to the *multiset* of elements of type τ. A multiset may be viewed as a function associating its multiplicity (a natural number) to each element. We use the following notation: $[\![]\!]$ for the empty multiset (of any type $\mathcal{M}(\tau)$), $[\![a_1,\ldots,a_p]\!]$ for a multiset with p occurrences of elements of type τ.

Example 77 (Multisets)**.** The multiset that contains the element A with the multiplicity 1, the element B with the multiplicity 2, and where all other elements are of multiplicity 0, is $[\![A,B,B]\!]$. Similarly, the multiset of face colors of the object from Figure 5.29a is $[\![$●,●,●,●,●$]\!]$ where each colored tablet directly encodes a particular color.

Signature Given a data type signature $\Omega = (S,F)$, the *signature with multisets* $\Omega^{\mathcal{M}}$ extends Ω in two steps. First, the set of type names is replaced by $S^{\mathcal{M}}$ defined as the set $S \cup \{\mathcal{M}(s) \mid s \in S\}$. Secondly, the profile mapping is considered as a function $\rho\colon F \to (S^\bullet)^* \times S^\bullet$.

Terms The definition of terms over a signature canonically extends to *terms over a signature with multisets* by considering that variables have a type in $S^{\mathcal{M}}$. The inductive construction of terms yields $T_{\Omega^{\mathcal{M}}}(X)$, the set of terms over the signature with multiset $\Omega^{\mathcal{M}}$.

Algebra Given a signature $\Omega = (S,F)$, an $\Omega^{\mathcal{M}}$*-algebra* $\mathcal{A}^{\mathcal{M}}$ extends an Ω-algebra by also considering carrier sets $\mathcal{A}_{\mathcal{M}(s)}$ for s in S such that $A_{\mathcal{M}(s)} = \{[\![a_1,\ldots,a_p]\!] \mid 0 \leq p, \forall k \in 0..p, a_k \in A_s\}$. The set of functions is also extended to match the profile of the function names in the signature with multisets.

In the sequel, we will provide signatures but always work with the corresponding signatures with multisets. Therefore, the terms and algebras will be considered over the signature with multisets, but we will omit the notation $\mathcal{M}$ everywhere except for the type names. Besides, we will often leave the algebra $\mathcal{A}$ implicit. When needed, the carrier set $\mathcal{A}_\tau$ of a data type τ will be written $\lfloor\tau\rfloor$.

By offering a concise and straightforward notation for multisets, our presentation of data types minimally meets our needs. Multiset properties such as commutativity of the element insertion or specification of the element multiplicity are semantically imposed. Note that multisets could also have been introduced by considering a higher-order approach as a starting point. We could have defined type constructions such as lists, sets, or tuples of elements of the same type. However, the resulting data types would have been more expressive than necessary, creating types such as sets of sets of elements when we only need a generic construction for denoting multisets of basic-type elements.

For dedicated embeddings, the user is expected to provide both a data type signature $\Omega = (S,F)$ and an Ω-algebra $\mathcal{A}$, with all the data types and functions required by its application domain to define its modeling operations.

Example 78 (Positions and colors data types with multisets)**.** We embed 2-Gmaps with positions on the vertices and colors on the faces via the signature $\Omega(pos,col) = (S(pos,col), F(pos,col))$ and the algebra $\mathcal{A}(pos,col)$.

The signature and algebra extend the ones given in Example 65 with multiset. We add the function names `center` and `mix` with the following profiles:

- `center` : $\mathcal{M}$(`point2D`) → `point2D`,

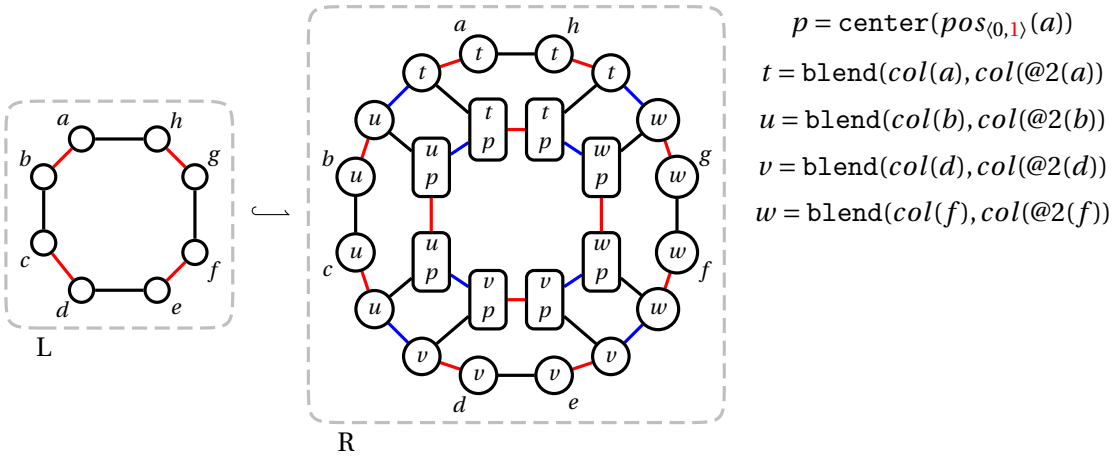


Figure 5.31: Rule for the barycentric face triangulation.

- $\texttt{mix} : \mathscr{M}(\texttt{colorRGB}) \to \texttt{colorRGB}$.

In $\mathscr{A}(pos, col)$, the function names are associated with the following functions:

- $\texttt{center}^{\mathscr{A}(pos,col)}$ corresponds to the function $center$, computing the barycenter of a multiset of points,
- $\texttt{mix}^{\mathscr{A}(pos,col)}$ corresponds to the function mix, defining the average color of a multiset of colors.

Note that we use multiset types for computations but never for attribution. At this point, multisets can therefore only be used as global variables (see Section 5.1.2), similar to the vector $\vec{v}$ in the vertex translation. We now discuss how to retrieve (multisets of) embedding values through the topological structure.

5.5.3 Orbit collects

With multisets, we can now write modeling operations more easily. For instance, Figure 5.31 provides the rule for the barycentric triangulation of a square face illustrated in Figure 5.27.

Each created triangle is colored by the mix between the triangulated face's original color and the one of its adjacent face, smoothing face colors. The terms t, u, v, and w encode the color smoothing via the function $\texttt{blend} : \texttt{colorRGB} \times \texttt{colorRGB} \to \texttt{colorRGB}$. In these terms, we use identifier terms as variables. The terms $col(a)$, $col(b)$, $col(d)$, and $col(f)$ are similar to the term $pos(a)$ used for the vertex translation in Figure 5.30b. They retrieve the colors of the nodes respectively mapped from a, b, d, and f in the modified Gmap, i.e., via the match. The terms $col(@2(a))$, $col(@2(b))$, $col(@2(d))$, and $col(@2(f))$ exploit traversals introduced in Section 5.5.1 to retrieve the color of the adjacent faces not present in the rule. Note that the terms stay well-defined with boundary edges. Indeed, a node x in a boundary edge will have a 2 loop, meaning that we would obtain $@2(x) = x$.

For the positions, we assume that the pre-existing vertices of the triangulated face have the same position after the operation. Thus, nodes a, b, c, d, e, f, g, and h do not have a term of type pos in R. However, we need to give a position to the newly created vertex, i.e., to the six nodes created on the right-hand side. Commonly, this created vertex is located at the barycenter of the

triangulated face. For instance, the triangulation of the top triangle in Figure 5.29a should add a vertex positioned at the barycenter of E, F, G, and H. Thus, we need to fetch the positions of E, F, G, and H. In other words, we need to retrieve the positions of all vertices of the face we are trying to triangulate. This retrieval is expressed by the identifier term $pos_{\langle 0,1\rangle}(a)$ in Figure 5.31. This term is intended to collect the multiset of positions in the $\langle 0,1\rangle$-orbit incident to the node identified by a. Assuming the same condition as in Example 76, where the variable a is mapped to the node a in Figure 5.29, $pos_{\langle 0,1\rangle}(a)$ collects the multiset $[\![\mathrm{E},\mathrm{F},\mathrm{G},\mathrm{H}]\!]$.

Intuitively, these collect functions are based on the orbit equivalence (Definition 70) that associates each embedding orbit to a single equivalence class and, therefore, to a single embedding value. Consequently, the multiplicity in the multiset does not depend on the orbit sizes but depends on the number of embedding orbits sharing the same value. In the case of the position embedding, each position value appears only in a single $\langle 0,1\rangle$-orbit because we do not want two vertices to coincide. Thus, any collection of position values would result in a multiset having each position at most once. For instance, in Figure 5.29, E, F, G, and H appear twice in $\langle 0,1\rangle(a)$, but the term $pos_{\langle 0,1\rangle}(a)$ only collect each position once. However, for most kinds of data, such as colors, quantities, or densities, one value can appear multiple times in the modeled object. For instance, the term $col_{\langle 0,1,2\rangle}(a)$ collects the face colors of the whole connected component, i.e., $[\![$●, ●, ●, ●, ●$]\!]$.

More generally, for all embeddings π and for all orbit types $\langle o\rangle$, we can consider a function name $\pi_{\langle o\rangle}$ on identifiers with profile $\rho(\pi_{\langle o'\rangle}) = \mathtt{id} \to \mathscr{M}(\tau_\pi)$. Thus, we extend the algebra $\mathscr{A}(\mathrm{G})$ for a Gmap G with functions $\pi^{\mathscr{A}(\mathrm{G})}_{\langle o\rangle}$. For a node v, $\pi_{\langle o\rangle}(id(v))$ collects one embedding value for each $\langle o_\pi\rangle$-orbit in $\mathrm{G}\langle o\rangle(v)$ and stores the collected values in a multiset. The multiset corresponds to the multiset of embedding values from the $\equiv_\pi$-equivalence classes of the nodes in the orbit $\langle o\rangle(v)$, i.e.:

$$[\![\pi([u]_\pi) \mid [u]_\pi \in \{[u]_\pi \mid u \in \langle o\rangle(v)\}]\!].$$

In other words, the multiset contains one value per $\langle o_\pi\rangle$-orbit in the $\langle o\rangle$-orbit adjacent to v. The function is well-defined on embedded graphs thanks to the unique existence of embedding values and the orbit consistency constraints.

Note that the extension with multisets from Section 5.5.2 occurs both in the algebra $\mathscr{A}(\Pi)$ used to manipulate Gmaps and in the algebra $\mathscr{A}(\mathrm{G})$ specified to a Gmap G. However, the collection function extension only concerns the algebra $\mathscr{A}(\mathrm{G})$. We then obtain identifier terms of types $\mathscr{M}(\tau_\pi)$ that can be used as variables in rules.

To sum up, identifiers from `id` allow traversals of the structure to either access embedding values of other nodes or collect values within an orbit. The trick comes from using identifier terms as variables in 'regular' terms. These identifier terms encompass the embedding access operators π, the neighbor access operators `@i`, and the collect operator $\pi_{\langle o\rangle}$. Thus, we obtain traversals depending on the modified Gmap, but we keep match morphisms with algebra morphism independent of the modified graph. Before moving on to the more practical aspects of this dissertation, we briefly compare the construction of embeddings via labels and attributes.

5.6 Embeddings via labels and embedding via attributes

Compared to our publication [3], we presented the addition of embeddings to the topological structure via attributes instead of labels. We obtain the same results and contributions, although the way to obtain them slightly differs.

Compared to attributes, labels inherently guarantee that a node has at most one embedding value. Besides, the algebraic computations are further decorrelated from the graph structure, simplifying the construction of a type using identifiers. Indeed, we can then reuse the set of nodes as a carrier set in the algebra without risk.

However, manipulating values with rules means that only one value can natively be considered for each node, meaning that the manipulation of multiple embeddings on the same Gmap may not be safe. Furthermore, handling values via labels means that rules are decorated with values rather than terms. Thus, abstract rules decorated with terms must be first transformed into rules with labels before applying the rule (with the eventual topological extension and embedding propagation). By contrast, using attributes naturally allows for the evaluation of terms via morphisms, which reduces the number of intermediate steps required to perform a transformation.

Summary of the chapter's contributions

In this chapter, we extended the formalization of the topological model and its rewriting with geometric information. In topology-based geometric modeling, such information is described by embedding the topological structure, i.e., the cells, in a geometric space. Within the theory of algebraic graph rewriting, we defined embedding via node attribution for the model of Gmaps. We then obtain embedded Gmaps defined by superimposing geometric constraints to the topological ones presented in the previous chapters. These constraints yield conditions on attributed rules to ensure consistency preservation for the geometry.

However, these conditions render cumbersome the writing of consistent operations. Thus, we proposed an automatic completion mechanism based on a topological extension adding the necessary elements to the rules and an embedding propagation ensuring that all geometric values are adequately modified. We proved the soundness of the application pipeline with respect to the geometric consistency and illustrated that this completion mechanism indeed simplifies the rules.

Although this completion mechanism offers a solution to modify embedding values without fully specifying the underlying concrete topology, it does not suffice to encapsulate geometric modifications. Therefore, we introduced a signature to describe the traversals of the modified structure. This signature yields an algebra where geometric computations can be realized on the structure of the modified graph before its transformation, which is not natively available with attributed rules. We call identifier terms the terms on this signature and use them as variables for regular rules. Thus, such that no additional consistency condition should be checked. We also added multisets to the signature and algebra to express computations acting on an undetermined number of values and extended the identifier terms with collect operators.

This chapter was relatively dense in content, but its ambition was rather simple: obtain a sound manipulation of the geometry. The most challenging part was to disentangle the geometry from the topology to allow for the representation of modeling operations independent from the underlying geometry while enabling attribute computation based on the structure of the modified graph. This challenge led us to introduce ad hoc solutions, namely a completion mechanism and specific terms mimicking structural traversals. We will discuss the practical integration of the results presented here with the rule schemes of Chapter 4 in Chapters 6 and 8.

Part II

Inference of geometric modeling operations with Jerboa

Chapter 6

Jerboa: a rule-based modeler generator

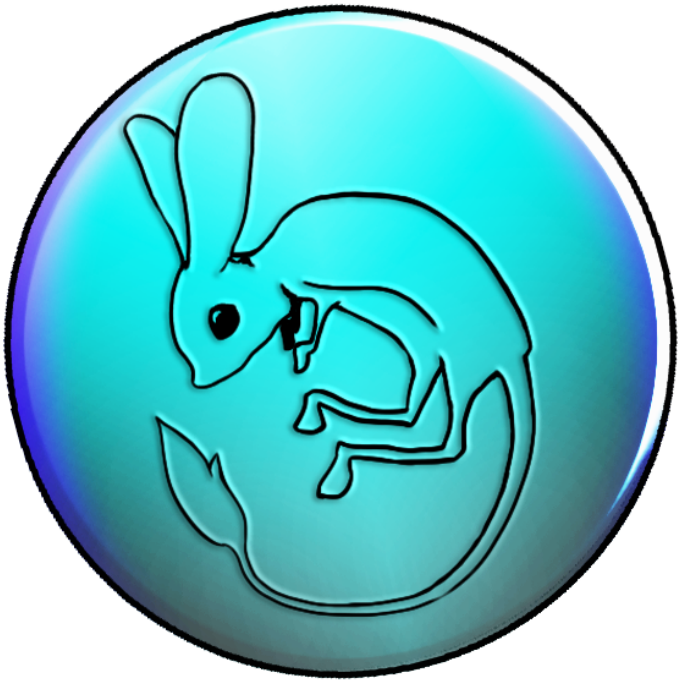


Figure 6.1: Jerboas are long-tailed hopping rodents from the deserts and steppes of eastern Europe, Asia, and northern Africa; Jerboa is also a rule-based modeler generator.

Personal note on the chapter

A side effect of manipulating graphs with the theory of categories is that graphs are always considered up to isomorphism. In particular, the result of a derivation is unique up to isomorphism. However, when leaving the world of abstraction and formalization that is category theory for more practical considerations where graphs are stored somewhere in memory, considering the result of a modeling operation up to isomorphism may seem bewildering. In practice, any such isomorphic graph is a correct computation. In this chapter, we voluntarily take distance from categorical constructions and definitions with two goals in mind. First, we seek to provide a construction of rule schemes understandable without extensive knowledge from more theoretical domains of computer science. Secondly, we wish to close the gap between the formalization and the implementation of our formal framework.

Contents

In this chapter, we present Jerboa, a framework for the design of geometric modelers. Jerboa has been introduced in [12] and uses Gmaps to represent objects and graph transformation rules to describe modeling operations. The design of a modeler requires specifying the set of applicable operations, i.e., the rule schemes, together with the geometric computations, i.e., the user signature from Chapter 5. While presenting Jerboa, we will also rephrase some constructions presented in the first part of this dissertation, but within the scope of Jerboa, e.g., only Gmaps are considered. We also narrow the discussion to the structures manipulated in Jerboa to explain how they are implemented. Besides, we use this opportunity to elucidate Gmap rewriting in non-categorical terms. We seek to ensure that all notions needed in the second part of this dissertation are effectively understood.

This chapter starts with an overview of Jerboa in Section 6.1. We present the different components of the library and how to generate a modeler with Jerboa. In Section 6.2, we characterize Gmaps as undirected graphs with arc labels in a more algorithmic perspective. We explain how topological cells can be manipulated via orbits and how to retrieve these orbits. We also provide insights into the representation of Gmaps and their embeddings in Jerboa. Once Gmaps are properly defined, we clarify how Jerboa handles modeling operations in Section 6.3 In particular, we present orbit-decorated rule schemes and their instantiation. This approach, which coincides with the representation of operations in Jerboa, is compared to the rule schemes introduced in the previous chapter. We recall the extension of embeddings from rules to rules schemes as introduced in [14] and sketch a description of Jerboa's rule application engine. In Section 6.4 we present some academic applications done with Jerboa. Lastly, we discuss some limitations of working with rule schemes in Section 6.5.

6.1 Jerboa

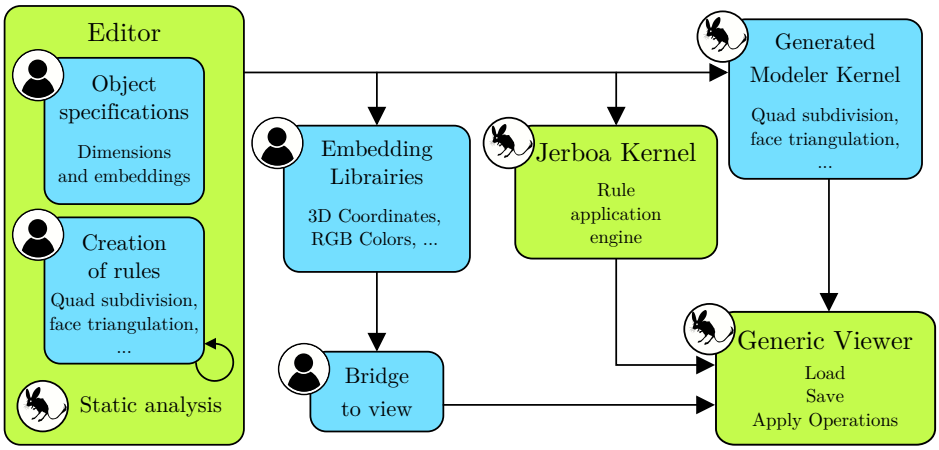


Figure 6.2: Jerboa's general architecture.

Jerboa [12] is a topology-based geometric modeling platform written in JAVA. The platform allows for the design of geometric modelers, i.e., software dedicated to geometric modeling. In the formal part of this dissertation, we mostly focused on the representation of Gmaps and Omaps as constrained graphs to provide a sound rewriting mechanism for the design of geometric modeling operations. However, geometric modelers require additional components. Some of these components are readily available in Jerboa or can be automatically generated by the platform, while

others require user interaction. The platform includes:

- a modeler editor equipped with verification tools to define the specific object model and the transformation rules describing the modeling operations,
- a rule application kernel to apply the modeling operations by rule derivation,
- a generic extensible viewer to visualize and manipulate the objects.

The general architecture is presented in Figure 6.2, where the Jerboa pictogram stipulates the automated parts, while the character pictogram indicates that user inputs are expected. The blue elements depend on the specific modeler being generated and, therefore, might vary between application domains. Conversely, the green elements are the main parts of the platform that can be used regardless of the application.

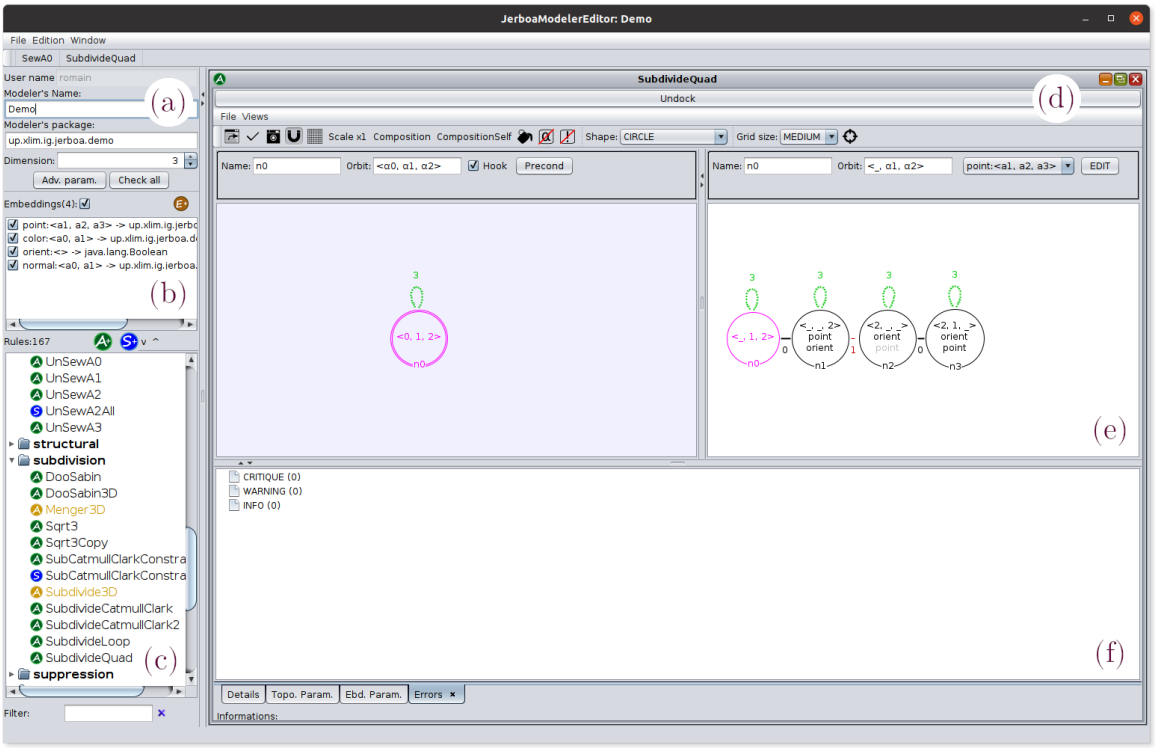


Figure 6.3: Jerboa's rule editor: (a) modeler's information, including the modeler's name and the dimension of the objects, (b) a list of the modeler's embeddings, (c) list of rules already defined in the modeler, (d) a window with edition tools for the design of rules, (e) left-hand side and right-hand side of the rule, (f) additional information about the rules, including the topological and geometric errors found.

We will present the different parts of the platform from the perspective of a user designing a modeler. The user starts with the modeler editor. They specify the characteristics of the manipulated objects. These characteristics are the dimension of the object, e.g., two, three, or forty-two, and the set of embedding used. Then, the user provides the embeddings needed for the given application. Jerboa offers some predefined embeddings, such as positions for the vertices or normals for the faces. For a user-defined embedding, the designer should provide a JAVA class describing all functions that can be applied to perform computations with the embedding. Then, the user writes the rules that encode the modeling operations suitable for the designed modeler. Although

the rules can be manually written in a text-based approach [16], Jerboa comes with a graphical rule editor within the modeler editor (see Figure 6.3). In the modeler editor, the syntax analyzer helps the user write the operations, highlighting the topological and geometric inconsistencies in the rule. Rules offer a formalized description of modeling operations, but the transformation of a geometric object requires effectively applying the rules. Therefore, Jerboa provides a rule application engine able to apply any rule written with the syntax of the rule editor.

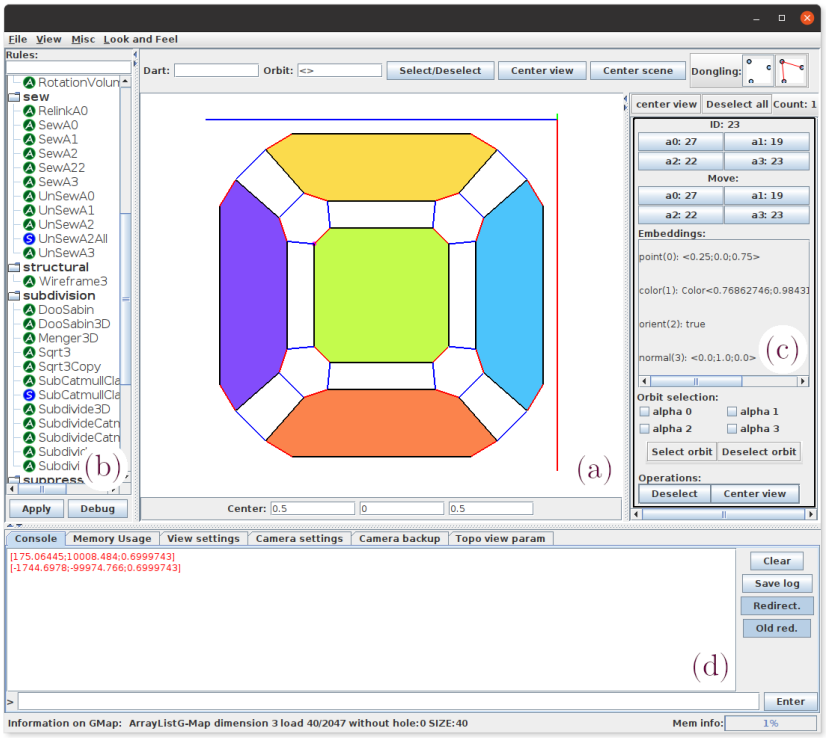


Figure 6.4: Jerboa's default viewer: (a) display of the geometric object being manipulated, (b) list of all rules defined in the modeler, (c) topological and geometric information about the selected darts, (d) panels to display additional settings, here the console, typically used for debugging or more complex interactions.

So far, we have a description of the topological model (Gmaps), its geometric properties (the embeddings), its modeling operations (the rules), and a solution to apply them (the rule engine). However, geometric modelers assume that objects can be interacted with and visualized. Therefore, Jerboa also provides a generic viewer (see Figure 6.4), which can either be used as-is, fine-tuned for a specific application, or replaced by a dedicated one. Note that in the screenshot of Jerboa's default viewer (Figure 6.4), the object is represented in what we call the *exploded view*. This exploded view displays the underlying Gmap instead of the geometric object. Each dart has a position computed as a barycentric explosion from its vertex position. The polygons described by the sequence of the darts in the (half) face are displayed as filled polygons to help visualize the (half) faces. The polygon's color coincides with the (half) face's color. The exploded view allows grasping both the topological and geometric information simultaneously.

With this last piece, the complete modeler can be generated. The generation converts each rule into a JAVA class, which, together with the embedding classes, Jerboa's rule application engine, and the viewer, are compiled to obtain executable code. By default, Jerboa is shipped[1] with

[1]Available in Lesson 3 of the Jerboa tutorial `http://xlim-sic.labo.univ-poitiers.fr/jerboa/doc/lesson-3-modify-and-write-a-first-rule/` (last accessed on August 2nd, 2022).

a 3D modeler, i.e., where objects are represented with 3-Gmaps. This modeler supports four embedding classes: a position embedding attached to the vertices (orbit $\langle 1,2,3\rangle$), a color embedding attached to the half-faces (orbit $\langle 0,1\rangle$), an orientation embedding attached to the darts (orbit $\langle\rangle$), and a normal embedding attached to the half-faces (orbit $\langle 0,1\rangle$). These correspond to the embeddings in Figure 6.3.

6.2 Implementation and manipulation of Gmaps

Recall from Chapter 1 that an n-generalized map encodes a subdivision of an n-dimensional space. In its combinatorial construction, a generalized map corresponds to a set of involutions all acting on the same set of darts with the property that the composition of some pairs of involutions yields an involution. In Chapter 3, we proposed a grah-based adaptation of n-generalized maps, which we called n-Gmap (see Def 32 in Section 3.2.2). A Gmap is defined as a topological graph, i.e., a graph arc-labeled by dimensions, subject to three topological constraints. In this section, we discuss the manipulation of Gmaps and their orbits in a lightweight approach, focusing on the representation of geometric objects.

6.2.1 Vocabulary

First, we want to clarify the use of some overlapping concepts, to help the reader distinguish between the different considerations, i.e., graph rewriting and geometric modeling.

1. We call *vertex* a 0-cell and *edge* a 1-cell. For instance, we can talk about the vertices and edges of the objects from Figures 6.5a and 6.6a.

2. We reserve the terms *node* and *arc* to refer to the constitutive elements of a graph, in agreement with algebraic graph rewriting [156, 57] used to model operations.

3. We call *darts* and *links* the elements of a generalized map, following the combinatorial definition of [43]. Therefore, we say that the representation of the stacked cubes with a Gmap in Figure 6.5e contains 96 darts, 48 0-links, 48 1-links, 48 2-links, and 88 3-links.

More generally, we will use the terms 'dart' and 'link' to refer to the elements of topological graphs, i.e., graphs arc-labeled by dimension. A reader knowledgeable in graph transformation will understand that we call darts the nodes of graphs in $\mathbb{D}$-**Graph** and links its arcs. We stick to the combinatorial vocabulary (3) in order to clarify that the Gmap represents a geometric object. Therefore, we can use the categorical vocabulary to refer to the elements of a rule scheme. We will preserve this vocabulary throughout the following chapters to better indicate which point of view should be considered.

With this refined vocabulary, an n-Gmap is an undirected graph with parallel links and loops, labeled on its links with dimensions in $0..n$, such that any dart has a unique incident i-link[2] per dimension and such that any $ijij$-path with $i+2 \leqslant j$ is a cycle. For the second part of this dissertation, all graphs will be considered undirected. Undirected graphs can be represented with a set of nodes V and a set of arcs E, where each arc is a set of either one or two vertices. In the former

[2] Since the graph is undirected, a dart is no longer the source, or target of an i-link, but simply have an incident i-link.

case, the arc is a loop. Note that this construction matches the categorical definition of undirected graphs (Definition 18 in Chapter 2). For convenience, we leave the incidence function implicit and write $G = (V_G, E_G)$ for an undirected graph with nodes V_G and arcs E_G. The subscript G may be omitted. Exploiting the link labels, we write $u \overset{i}{\bullet\!\!-\!\!\bullet} v$ when two darts of a topological graph G share a i-link, i.e., where the arc $\{u, v\}$ is labeled by i. Following the color code used so far, we also write $u \bullet\!\!-\!\!\bullet v$ for a 0-link, $u \bullet\!\!=\!\!\bullet v$ for a 1-link, $u \bullet\!\!\equiv\!\!\bullet v$ for a 2-link, and $u \bullet\!\!\equiv\!\!\bullet v$ for a 3-link. We call v the i-neighbor of u when $G \models \mathtt{I}_G(i)$, i.e., when all darts have a unique incident i-link (Definition 29 in Chapter 3).

6.2.2 Representation of Gmaps

In Section 3.2.2 (Chapter 3), we explained how to represent an object with a Gmap through its cellular decomposition. The decomposition was done in 2D but holds for higher dimensions, as illustrated in Figure 6.5.

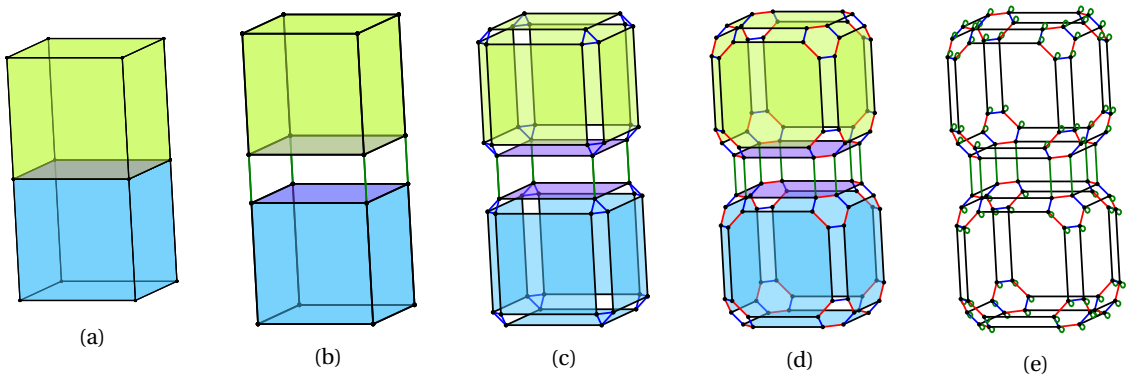


Figure 6.5: Cellular decomposition of two stacked cubes: (a) two cubes sharing a face, (b) split on dimension 3, (c) dimension 2, (d) dimension 1, and (e) dimension 0, which yields the corresponding 3-Gmap after adding the missing loops.

We now present a counterpart construction, i.e., a bottom-up approach where we start from the darts of the object and then aggregate them recursively. We call this process *dimensional unification*, which provides a different intuition about the representation of geometric objects with Gmaps.

The object in Figure 6.6a corresponds to the object used for the cellular decomposition in Chapter 3. For dimension unification, we start by creating all the necessary darts. Each dart uniquely encodes a tuple of topological cells. Since we are representing a 2D object, each dart encodes a tuple (vertex, edge, face), and we add one dart for each of these tuples. Note that all tuples are not to be considered, only those where the vertex is incident to the edge and the edge to the face. Loosely said, a dart represents the vertex part of the edge part of a face. The addition of all darts is depicted in Figure 6.6b, where the darts are positioned close to their vertex, close to their edge, and inside their face. Next, we need to explain how the darts are connected. The darts' connections describe the relations between the topological cells. Working by increasing dimensions, we first add the 0-links. A 0-link signifies that the darts encode the same topological cells except for the 0 dimension. In other words, we add a 0-link between two darts that belong to the same edge and the same face while being in distinct vertices. The addition of the 0-link is depicted in Figure 6.6c. We now repeat this construction by increasing dimensions. We add the 1-links between

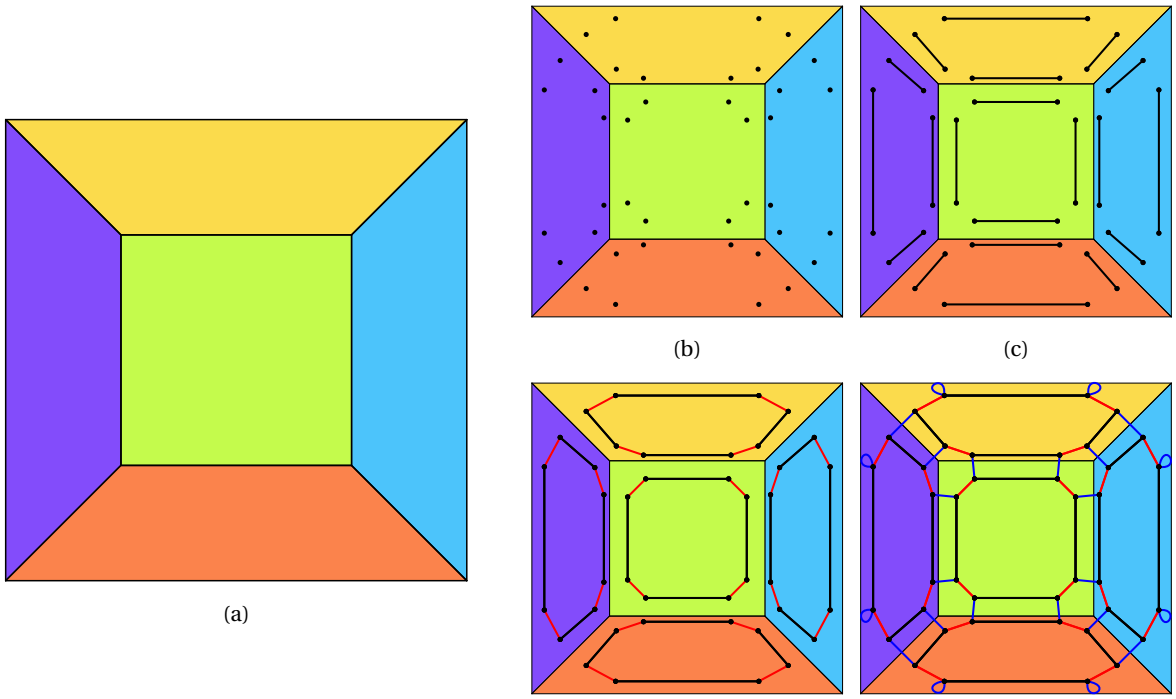


Figure 6.6: Dimensional unification of a surface: (a) a surface composed of five quads, (b) addition of the darts for the tuples (vertex, edge, face), (c) unification along 0, (d) unification along 1, and (e) unification along 2, which yields the corresponding 2-Gmap.

darts that belong to the same vertex and the same face but belong to distinct edges, as illustrated in Figure 6.6d. Finally, we add the 2-links between darts in the same vertex and edge but in distinct faces, as shown in Figure 6.6e. At any time during the unification procedure, a loop is added if a dart cannot be i-linked because its i-cell belongs to the object's boundary. The (quasi-)manifold property of the represented objects ensures that the construction yields a valid Gmap.

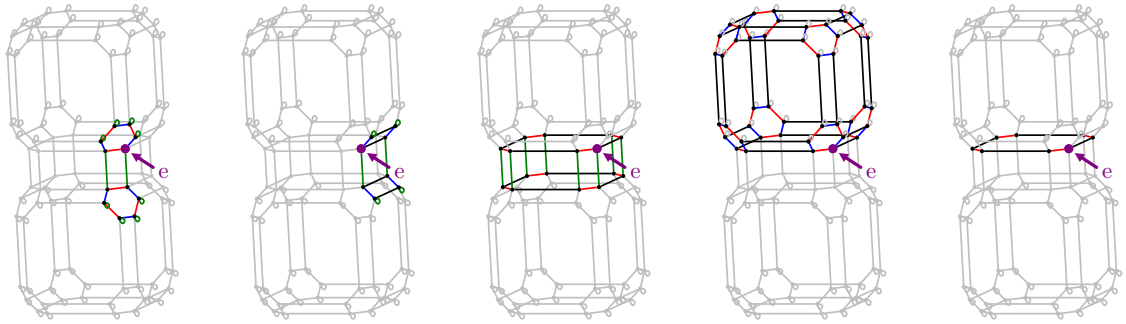
On a side note, I developed a tool for constructing the representation of objects with Gmaps by superimposing the topological structure over the object, i.e., as in Figure 6.6e. This representation offers a different intuition when visualizing surfaces since the user can see the exact object while grasping the internal topological relations. This tool is integrated within Jerboa and can be used for other projects. Note that all images of objects or Gmaps provided in the second part of this dissertation have been generated using Jerboa, some of them with this new tool.

In Jerboa, Gmaps are represented as adjacency lists. Recall that each dart has a unique incident i-link per dimension. Therefore, each dart only needs to store its i-neighbor for each dimension i. Thus, the connectivity of the darts can be stored via references in an array. Given a dart d, we can then access its i-neighbor via `d@i`. Besides its i-neighbors, each dart also stores a unique identifier such that a Gmap can then be described as a collection of darts.

6.2.3 Topological cells as orbits

In Chapter 3, we explained that Gmaps do not explicitly store the cell subdivision of a geometric object. We discussed how to retrieve the topological cells in Example 33 (Section 3.2.2). In Chapter 5, we discussed the representation of topological cells using subgraphs called orbits (Definition 56), which is only valid for Gmaps. In the second part of this dissertation, we will work

with Jerboa and, therefore, with an orbit description of topological cells. We recall that an *orbit* corresponds to a subgraph induced by all the darts reachable from an initial dart, only using links from a subset $\langle o \rangle$ of $0..n$. The subgraph is written $G\langle o \rangle(v)$ or $\langle o \rangle(v)$, while $\langle o \rangle$ is the *orbit type* and $G\langle o \rangle(v)$ is said to be an $\langle o \rangle$-orbit. When only the type of the orbit is given, we may need to specify that it is an n-orbit type to signify that the orbits should be considered in a $(0..n)$-topological graph.



(a) (b) (c) (d) (e)

Figure 6.7: Orbits incident to the purple dart e in the Gmap of Figure 6.5e: (a) vertex $G\langle 1,2,3\rangle(e)$, (b) edge $G\langle 0,2,3\rangle(e)$, (c) face $G\langle 0,1,3\rangle(e)$, (d) volume $G\langle 0,1,2\rangle(e)$, and (e) face of volume $G\langle 0,1\rangle(e)$.

Although we only discussed orbits on 2D objects so far, the construction does not depend on the dimension. For instance, if we call G the Gmap of Figure 6.5e, we can describe the topological cells incident to the pointed purple dart e from Figure 6.7.

Vertex The 0-cell incident to dart e is given in Figure 6.7a. This cell contains the dart and every dart reachable by all links except 0-links (i.e., 1, 2, and 3-links), along with the links themselves. This subgraph is written $G\langle 1,2,3\rangle(e)$.

Edge The 1-cell incident to dart e is displayed in Figure 6.7b. This cell contains all darts and links gathered in the traversal of G with all the dimensions but 1 when starting from e. The subgraph $G\langle 0,2,3\rangle(e)$ corresponds to this edge.

Face The 2-cell $G\langle 0,1,3\rangle(e)$ incident to e is shown in Figure 6.7c.

Volume The 3-cell incident to e is the subgraph $G\langle 0,1,2\rangle(e)$ illustrated in Figure 6.7d.

Retrieving an orbit intuitively provides all darts of G that correspond to a common topological element. Such topological elements encompass cells and more restricted elements, such as faces of volumes or edges of faces (of volumes) in 3D and half-edges or corners of faces in 2D. For example, the orbit $G\langle 0,1\rangle(e)$ described in Figure 6.7e represents the face of one volume (also called half face) incident to e. Indeed, the face incident to e is the orbit $G\langle 0,1,3\rangle(e)$. Removing the dimension 3 splits the two half faces, one in the top cube and one in the bottom cube.Since the full graph G of Figure 6.5e contains a single connected component, it corresponds to the orbit $G\langle 0,1,2,3\rangle(e)$.

With this representation, we can retrieve an orbit with a graph traversal. Since each dart stores its neighboring relations, we encode an orbit as a set of darts. Algorithm 4 present a breadth-first search approach, but a depth-first search algorithm would also work. In this algorithm, the function `enqueue` adds an element at the end of a queue, while the function `dequeue` extracts the first element of the queue. The function `add` adds an element to a collection.

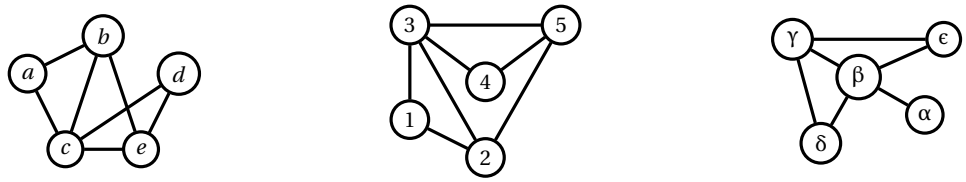
Algorithm 4: Orbit of a dart in a Gmap.

Input: A dart d in an n-Gmap G and an n-orbit type $\langle o \rangle$.
Output: The subset of all darts in G that belongs to $G\langle o \rangle(d)$.

```
1  Function orbit(⟨o⟩,d):
2      O ← ∅
3      Q ← empty queue
4      enqueue(Q,d)
5      while Q not empty do
6          v ← dequeue(Q)
7          if v ∉ O then
8              add(O, v)
9              foreach i ∈ ⟨o⟩ do
10                 enqueue(Q, v@i)
11     return O
```

In Jerboa, we often need to compare orbits to determine whether two orbits are isomorphic. Orbit comparison intervenes as a subroutine for applying rules. We next detail the computation of orbit isomorphism, even though we only detail the application of rules later in Section 6.3.

6.2.4 Isomorphism of orbits in a Gmap



(a) (b) (c)

Figure 6.8: Isomorphic and non-isomorphic graphs. The graphs (a) and (b) are isomorphic via the bijection on nodes $\{a \mapsto 1, b \mapsto 2, c \mapsto 3, d \mapsto 4, e \mapsto e\}$. The graph (c) has a node of degree[3] one (node α), while neither graph (a) nor graph (b) has a node with such a degree. Therefore, the graph (c) is not isomorphic to the graphs (a) and (b).

The graph isomorphism problem is to determine whether two finite graphs are isomorphic. For instance, the graph in Figure 6.8a is isomorphic to the graph in Figure 6.8b but not to the graph in Figure 6.8c. This problem is well-known in the graph rewriting community. Indeed, the standard approach to obtaining a match is to nondeterministically find a graph isomorphic to the rule's left-hand side in the host graph. For instance, [2] uses the *VF2* algorithm [34], which performs a comparison of graphs. In general, the graph isomorphism problem is not known to be solvable in polynomial time, nor to be NP-complete [70], although a $2^{O(\sqrt{n \log n})}$ has been proposed [5, 4]. However, many practical algorithms exist. Solutions based on a graph comparison directly compute an isomorphism between the two graphs. These approaches have the advantage that only one isomorphism needs to be computed but have the drawback that comparing many graphs may be computationally heavy. A second approach to obtaining practical graph isomorphism algorithms is to compute canonical labels. For a graph G, C(G) is a canonical label of G granted that for all graphs H, we have C(G) = C(H) if and only if G and H are isomorphic. Typical examples of this approach revolve around the computation of the minimal (or maximal) automorphism of G for a suitable ordering. See for instance the *nauty* algorithm [128].

Moreover, the graph isomorphism problem can be shown to be polynomial for some families of graphs. For instance, the graph isomorphism problem is polynomial for graphs of bounded valence [126]. In the case of combinatorial maps, the subisomorphism problem, which is to determine whether a graph admits a subgraph isomorphic to another graph, can be solved in polynomial time [41]. Therefore, the isomorphism problem is also polynomial. Note that in [41], the permutation-based representation is used, but the result holds in the graph framework. Besides, if we view Gmaps (or Omaps) as graphs, we understand that looking for an isomorphism between Gmaps or orbits in Gmaps is simply a joint traversal algorithm (see the discussion in [44]). Signatures [79] exploit the idea of finding the canonical label of a graph in the case of combinatorial maps. Computing these signatures is as costly as finding the isomorphism (requires time $O(|\mathrm{D}|^2)$ if D is the set of darts). However, comparing the signatures can then be realized in linear time, which can speed up the process when many comparisons need to be realized, e.g., in the query-replace framework presented in [44].

Algorithm 5: Orbit isomorphism.

Input: Two darts d and d' in an n-Gmap G and two n-orbit types $\langle o\rangle$ and $\langle o'\rangle$.
Output: A node mapping from $G\langle o\rangle(d)$ to $G\langle o'\rangle(d')$ if $G\langle o'\rangle(d)$ and $G\langle o'\rangle(d')$ are isomorphic, null otherwise.

```
1  Function isomorphic(⟨o⟩, d, ⟨o'⟩, d'):
2      mapd ← empty map
3      mapr ← empty map
4      Q ← empty queue
5      enqueue(Q,d)
6      put(mapd,d,d')
7      put(mapr,d',d)
8      while Q not empty do
9          v ← dequeue(Q)
10         v' ← get(mapd,v)                    // Dart associated with v in the isomorphism
11         foreach i ∈ ⟨o⟩ do
12             x ← v@i
13             j ← [⟨o⟩ ↦ ⟨o'⟩](i)             // Compute the image of i in the relabeling ⟨o⟩ ↦ ⟨o'⟩
14             x' ← v'@j
15             if x ∈ mapd and get(mapd,x) ≠ x' then   // Non-injectivity from G⟨o'⟩(d') to G⟨o⟩(d)
16                 return null
17             if x' ∈ mapr and get(mapr,x') ≠ x then   // Non-injectivity from G⟨o⟩(d) to G⟨o'⟩(d')
18                 return null
               // Keep building the mapping
19             put(mapd,x,x')
20             put(mapr,x',x)
21             enqueue(Q,x)
22     return mapd
```

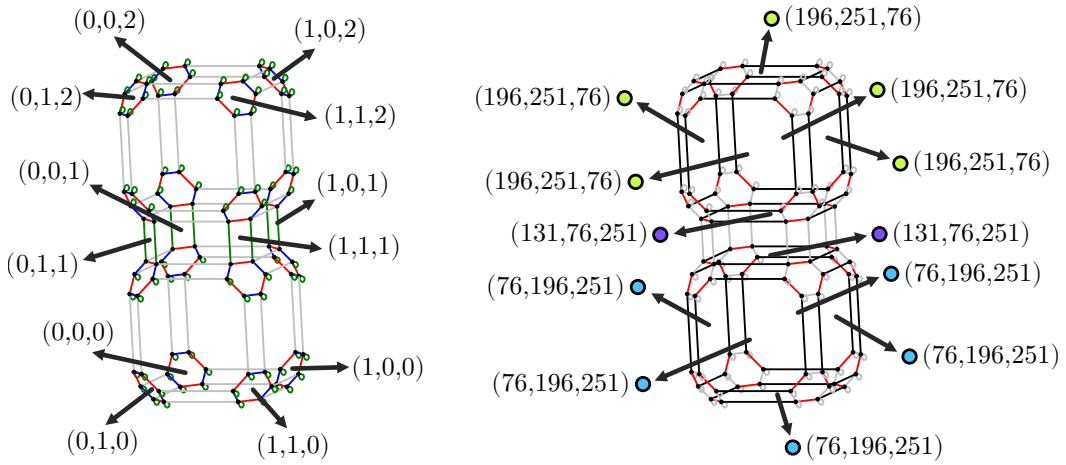
Orbit comparison occurs when applying a Jerboa rule scheme with several connected components, and the retrieved orbits need to be isomorphic for the rule to be applicable. Two additional constraints must be considered when determining whether two orbits are indeed isomorphic. First, the orbits are not given and need to be computed. Indeed, the rule application mechanism starts from a hook-to-dart mapping, not from the built orbits. Secondly, the isomorphism computation is realized up to the relabeling. Indeed, the hooks may have different orbit types. In such cases, a non-trivial relabeling function exists between the considered orbits. A joint graph traver-

sal (up to relabeling) can solve the orbit isomorphism problem, exploiting the incident arcs constraint. A breadth-first approach is presented in Algorithm 5. During the traversal, we build two hash maps `mapd` and `mapr`. The map `mapd` stores a node mapping from $G\langle o\rangle(d)$ to $G\langle o'\rangle(d')$, describing a partial injective function from $G\langle o\rangle(d)$ to $G\langle o'\rangle(d')$ (arcs included). Similarly, the map `mapr` stores a node mapping from $G\langle o'\rangle(d')$ to $G\langle o\rangle(d)$. As we traverse the graph, we ensure the injectivity of the two maps. If a non-injectivity can be detected, the orbits are non-isomorphic, and the algorithm can stop. If the traversal goes through, we obtain two inverse injective maps, meaning that the orbits are isomorphic. The function `put` adds a pair key-value to a (hash) map, while the function `get` retrieves the value associated with a key in the map. We also assume that we can check whether a key has a value in the map, written as the set membership relation $\in$.

In practice, multiple orbits may have to be tested for isomorphism (up to relabeling). In such cases, the algorithm can be extended by using pairs of maps between a reference orbit and all the other orbits.

6.2.5 Embedded Gmaps

In Chapter 5, we explained how the geometric data is handled on Gmaps via graph attributes. From a more practical perspective, embeddings can be viewed as functions [14]. The embedding functions map each topological cell to its relevant data, e.g., a vertex to a position and a half face to its color. This minimal embedding information allows for a simple display of objects, with edges represented as straight segments. An intuitive description of the two cubes' embedding is provided in Figure 6.9. In practice, each dart stores a reference to its value for each embedding, and we can access the value of an embedding π on a dart d via `d.π`.



(a) (b)

Figure 6.9: Embedded representation of the stacked cubes: (a) embedding of the vertices position with $pos\colon \langle 1,2,3\rangle \to$ `point3D`, both cubes are of length 1, (b) embedding of the faces color with $color\colon \langle 0,1\rangle \to$ `colorRGB`, the colors described by the RGB values are displayed next to the values.

6.3 Gmap rewriting

In the first part of this dissertation, we derived a formal framework from the graph-based representation of Gmaps to handle modeling operations as graph transformation rules. Such rules allow

matching some structure within a graph and replacing it with a different structure. DPO rules did not support a proper definition of modeling operations since they overspecified the transformation. Thus, we extended rules to rule schemes, parameterized by a set of words and instantiable via a functor once a pattern graph was chosen (see Chapter 4). Our instantiation functor relies on a categorical product of graphs. For the restricted case of Gmaps and orbit-based generalization, rule schemes and their instantiation can be explained using relabeling functions. We call Jerboa rule scheme the construction of a rule scheme that relies on relabeling functions to distinguish it from notions presented in the previous chapters. Once the complete framework has been built, we will discuss how it relates to that of Chapter 4.

In this framework, the rule is assigned an orbit type as a parameter, intuitively describing the rule's modified orbit. Each node is also assigned an orbit type used to compute the operation. In other words, graph schemes are not labeled with pairs on the arcs but have decorations[4] on nodes that encode the relabeling functions [145, 144]. The simplest construction to understand the application of an operation described with a rule scheme is probably that of [14], revisited in [138], which we will present in this section.

6.3.1 A folded representation of modeling operations

Figure 6.10 provides two possibilities for the vertex insertion in a 2-Gmap, based on the freedom of the edge. A free edge is a $\langle 0,2\rangle$-orbit where the 2-links are loops, while a sewn edge is a $\langle 0,2\rangle$-orbit where the 2-links are non-loop arcs. This distinction gives rise to two configurations for the vertex insertion operation: one for a free edge (see Figure 6.10a for the rule and Figure 6.10c for an example of an application) and one for a sewn edge (see Figures 6.10b and 6.10d). These figures display a zoom on the modified part where the Gmap and the object are drawn to ease understanding. In particular, the object's vertices are marked with dots to highlight the added vertices.

The incident arcs property of Gmaps ensures that the choice of a single dart is sufficient to apply these rules. We can build the complete mapping by a joint traversal of the left-hand side and the rewritten graph. We illustrate the construction of the match with the rule application of Figure 6.10c. This application assumes that the match maps node x onto node a. Because a match has to preserve the node adjacency and the labels, the only possible match maps the arcs incident to x onto the arcs incident to a. Thus, $x \bullet\!\!-\!\!\bullet x$ is mapped onto $a \bullet\!\!-\!\!\bullet a$, while $x \bullet\!\!-\!\!\bullet y$ is mapped onto $a \bullet\!\!-\!\!\bullet b$. Now, mapping the arcs incident to x provides information on how the match should map the nodes adjacent to x. Indeed, a valid match that maps $x \bullet\!\!-\!\!\bullet y$ onto $a \bullet\!\!-\!\!\bullet b$ must map y onto b. By recursively exploring the arcs incident to the newly found nodes, we can unambiguously recreate the match from the sole information that x is mapped onto a. Intuitively, the incident arcs constraint allows reconstructing the match from a partial mapping. The minimal information required is the mapping of one node per connected component of L. We retrieve the notion of hook defined for rule schemes in Chapter4, but in the case of simple rules.

A Jerboa rule scheme describes a folded representation of a transformation, which must be unfolded to obtain the actual graph transformation that can modify an object. For instance, the two configurations of the vertex insertion can be unified by folding the edge along its 2-links. We

[4] We use the term decoration at the information associated with the nodes are neither labels nor attributes as defined in graph transformation. In [12], these decorations are referred to as variables, following the construction of [101].

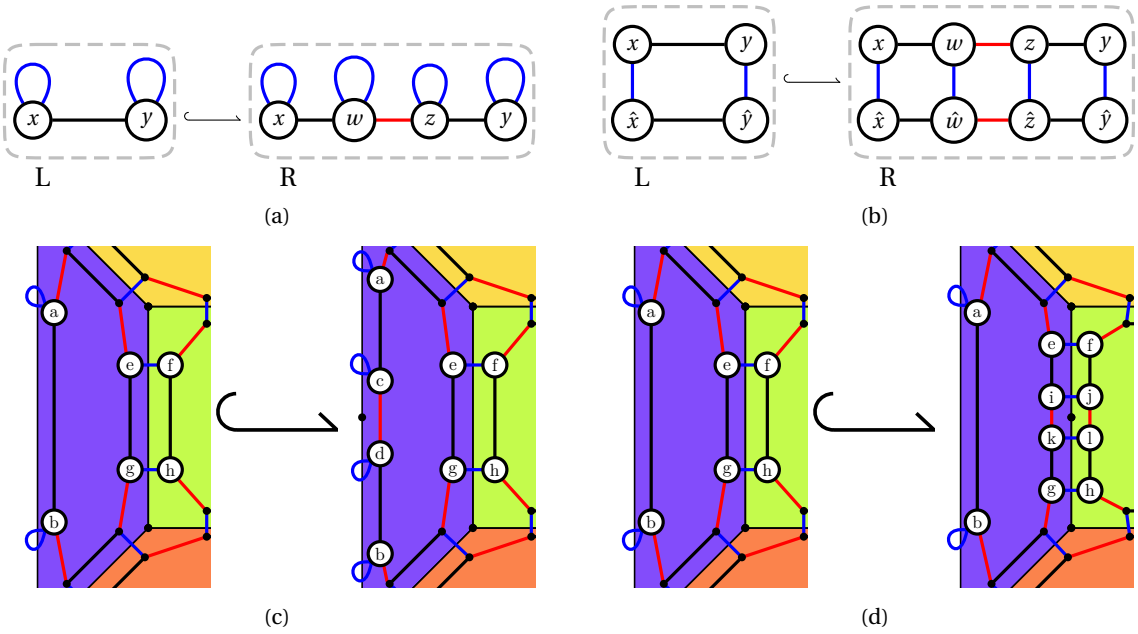


Figure 6.10: Graph transformation rules for the vertex insertion on the object of Figure 6.6 Graph transformation rule for the vertex insertion in a free edge (a) and its application on a 2-Gmap on an outer edge (c) via the match deduced from $x \mapsto a$. Graph transformation rule for the vertex insertion in a sewn edge (b) and its application on a 2-Gmap on an inner edge (d) via the match deduced from $x \mapsto e$.

obtain the rule of Figure 6.11a. This rule is parameterized with the orbit type $\langle 2 \rangle$. To obtain a graph-level rule, we choose a graph that consists of an orbit $\langle 2 \rangle$. This graph is then used to unfold all the node decorations. If unfolded as a 2-loop, the Jerboa rule scheme provides the rule of Figure 6.10a, while unfolding it with two darts sharing a 2-link yields the rule of Figure 6.10b. We can even further fold the rule to obtain the Jerboa rule scheme of Figure 6.11b. These folding and unfolding constructions are defined via relabeling functions.

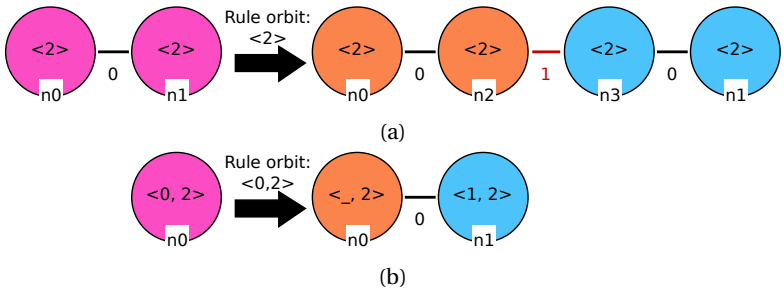


Figure 6.11: Jerboa rule schemes for the vertex insertion: (a) by folding the 2-links and (b) both the 0- and 2-links.

Relabeling Function

Relabeling functions allow rewriting orbit types.

Definition 86 (Relabeling function [138])**.** *A* relabeling function *of dimension n, or relabeling function, is a partial function $f : 0..n \to 0..n \cup \{\_\}$, injective on $0..n$, where '_' is a special symbol called* removing symbol.

The application of a relabeling function to an orbit type is its application to each dimension within the orbit. For example, $\{0 \mapsto 1, 2 \mapsto 2\}(\langle 0,2 \rangle) = \langle 1,2 \rangle$. The positions of the dimensions within

the orbit type entirely describe the relabeling function if we assume a reference orbit type $\langle o \rangle$ and a relabeled orbit type $\langle o' \rangle$. For instance, given $\langle 0,2 \rangle \mapsto \langle 1,2 \rangle$, one can unambiguously reconstruct the relabeling function $\{0 \mapsto 1, 2 \mapsto 2\}$. The motivation to use relabeling functions is to encode orbit rewriting. Therefore, we usually denote such a function as a relabeling of orbit types. Let $\langle o \rangle = (o_i)_{i \leq k}$ be the set of dimensions for which f is defined (ordered by increasing value), then f is written $\langle o \rangle \mapsto \langle (f(o_i))_{i \leq k} \rangle$. The injectivity property simply means that a dimension d cannot appear twice in $\langle o' \rangle = \langle (f(o_i))_{i \leq k} \rangle$. Let us remark that $\langle o' \rangle$ is not strictly speaking an orbit type, as it may contain the symbol '_'. In this sense, it is a *generalized orbit type*, which we will also call orbit type for convenience. Please note that the relabeling function's domain $\langle o \rangle$ must not contain the removing symbol. A relabeling function naturally extends from orbit type rewriting to orbit rewriting. Given a relabeling function $\langle o \rangle \mapsto \langle o' \rangle$ and an orbit $\langle o \rangle(v)$, one can build the orbit $\langle o' \rangle(v)$ by relabeling all links according to the function.

In Figure 6.12, we illustrate the application of relabeling functions to orbit graphs. The relabeling function $\{0 \mapsto 1, 2 \mapsto 2\}$ applied on the graphs of Figure 6.12a yields the graphs of Figure 6.12b. The highlighting on the graphs of Figure 6.12 will be exploited later. Here, we are only interested in the link relabeling, i.e., the modification of the link color. The 2-loops incident to nodes a and b yields 2-loops incident to nodes $a1$ and $b1$, as described by the relabeling $2 \mapsto 2$. Similarly, the relabeling $0 \mapsto 1$ transforms the link $a \bullet\!\!-\!\!\bullet b$ into $a1 \bullet\!\!-\!\!\bullet b1$. The application of the same function on the graphs of Figure 6.12d yields the graphs of Figure 6.12e.

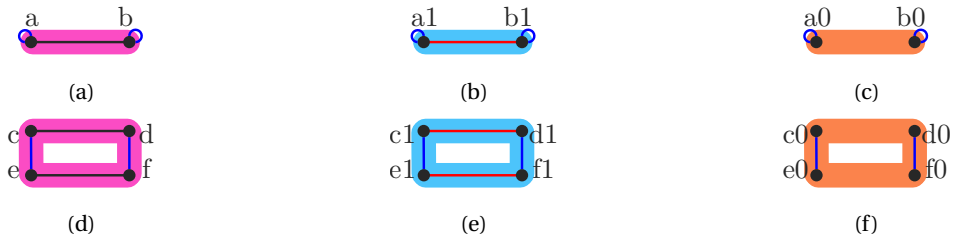


Figure 6.12: Relabeling functions applied to orbit graphs: orbits (a) and (d) of type $\langle 0,2 \rangle$, label modification (b) and (e) via the relabeling function $\langle 0,2 \rangle \mapsto \langle 1,2 \rangle$, and label deletion (c) and (f) via the relabeling function $\langle 0,2 \rangle \mapsto \langle \_,2 \rangle$.

As suggested by its name, the removing symbol '_' represents the deletion of the relabeled dimension, thus extending the definition of relabeling functions and their application to orbits. For instance, $\{0 \mapsto \_, 2 \mapsto 2\}$ denotes the removal of the 0-label while preserving 2. Similar to any dimension, the removing symbol may appear in the orbit type of a node decoration. For example, $\langle \_,2 \rangle$ is a valid node decoration. Assuming a reference orbit type $\langle 0,2 \rangle$, one can unambiguously reconstruct the relabeling function $\{0 \mapsto \_, 2 \mapsto 2\}$. When applied to an orbit, all links relabeled with _ are deleted. Two examples are provided in Figures 6.12c and 6.12f, using the graphs of Figure 6.12a and 6.12d as references. In these examples, the links $a \bullet\!\!-\!\!\bullet b$, $c \bullet\!\!-\!\!\bullet e$, and $d \bullet\!\!-\!\!\bullet f$ do not have corresponding links between $a0$ and $b0$, $c0$ and $e0$, and $d0$ and $f0$.

Relabeling functions allow encoding folded representation of graphs and rules as Jerboa graph schemes and Jerboa rule schemes.

Definition 87 (Jerboa graph scheme and rule scheme)**.** *Let $\langle o \rangle$ be an orbit type defined on $0..n$.*

A Jerboa graph scheme *of dimension n on $\langle o \rangle$, $(n, \langle o \rangle)$-Jerboa graph scheme, or simply Jerboa graph scheme, consists of a graph $\mathcal{G}$ whose arcs are labeled on $0..n$ and nodes are decorated with*

generalized orbit types of the same size as $\langle o \rangle$. *For each node* v *of* $\mathcal{G}$, *we write* $\langle o^v \rangle$ *the orbit type decorating* v.

A Jerboa rule scheme *on* $\langle o \rangle$, *or simply Jerboa rule scheme, is a rule* $\mathcal{L} \xleftarrow{\langle o \rangle} \mathcal{R}$ *where* $\mathcal{L}$ *and* $\mathcal{R}$ *are Jerboa graph schemes defined on* $\langle o \rangle$.

The orbit type of a Jerboa rule scheme is also said to be its parameter and might be omitted when the context is clear. Both Jerboa graph schemes of a Jerboa rule scheme must share the same orbit type. The node decorations of a Jerboa graph scheme are used as placeholders to encode any orbit of the given orbit type. More precisely, each node of a Jerboa graph scheme is intended to be substituted by an orbit whose type matches its decoration.

A Jerboa graph scheme reduced to a unique node decorated with the orbit type $\langle 0,2 \rangle$ describes either a free or sewn edge, as depicted in Figure 6.12a and 6.12d. Other examples of Jerboa graph schemes can be found throughout this section, for instance, the left-hand and right-hand sides of the rules of Figure 6.11, or the graphs of Figures 6.13a, and 6.14a.

As soon as the Jerboa graph scheme $\mathcal{G}$ contains several nodes, an additional condition comes into play. The size condition on the orbit types decorating the nodes of $\mathcal{G}$ means that they all share the same number of symbols as $\langle o \rangle$. Therefore, the node substitutions can be obtained from relabeling functions built with $\langle o \rangle$. All the nodes of $\mathcal{G}$ will be substituted based on the same orbit typed by $\langle o \rangle$, whose links will be relabeled by the relabeling function attached to the nodes.

Jerboa graph schemes can be used to define rules with the requirement that both the left-hand and right-hand sides are Jerboa graph schemes defined on the same orbit type.

Instantiation

In the case of the graph scheme and rule schemes of Chapter 4, the instantiation process relied on a functor defined with a suitable choice of pattern graph. This functor relies on a product of graphs in a category where arcs are labeled by pairs that simulate a relabeling. When exploiting Jerboa rule schemes, we can explain the construction using the relabeling functions we just introduced. The choice of a suitable pattern graph is replaced with the choice of a suitable orbit graph. Unfolding a Jerboa graph scheme through relabeling functions, i.e., instantiating it, is defined separately for nodes and arcs.

For the following definitions, we consider a Jerboa graph scheme $\mathcal{G}$ defined on the orbit type $\langle o \rangle$ and a graph O that consists of an orbit typed by $\langle o \rangle$. The goal is to define the graph $\iota^{\langle o \rangle}(\mathcal{G}, \mathrm{O})$ which corresponds to the instantiation of $\mathcal{G}$ with O. We provide a mathematical description of the instantiation process for reasoning in Chapter 7.

Nodes The first component is the instantiation of a node s of $\mathcal{G}$. In such a case, the instantiation process is reduced to applying the relabeling function $\langle o \rangle \mapsto \langle o^s \rangle$ to the orbit graph O chosen for the instantiation. We can then instantiate all the nodes by applying each relabeling function to copies of O.

Figure 6.13a provides the two nodes of the right-hand side of the Jerboa rule scheme from Figure 6.11b. We consider the instantiation on the orbit $\langle o \rangle = \langle 0,2 \rangle$.

Since node $n0$ has the orbit type $\langle \_,2 \rangle$, the relabeling function $\langle o \rangle \mapsto \langle o^{n0} \rangle$ is the function $\{0 \mapsto \_, 2 \mapsto 2\}$. Similarly, node $n1$ has the orbit type $\langle 1,2 \rangle$ which yields the relabeling function $\langle o \rangle \mapsto \langle o^{n1} \rangle$ given by $\{0 \mapsto 1, 2 \mapsto 2\}$. These two functions correspond to the relabeling function already

Figure 6.13: Instantiating the nodes of a Jerboa graph scheme: (a) discrete Jerboa graph scheme, (b) instantiation with the orbit graph of Figure 6.12a, and (c) instantiation with the orbit graph of Figure 6.12d.

illustrated in Figure 6.12. Thus, we obtain the instantiations for a free edge or a sewn edge by taking the union of the graphs. These instantiations are respectively given in Figures 6.13b and 6.13c.

Definition 88 (Instantiation of the nodes)**.** *If v is a node of $\mathcal{G}$, its instantiation with* O *is the graph obtained by the application of the relabeling function $\langle o\rangle \mapsto \langle o^v\rangle$ to* O*:*

$$\iota^{\langle o\rangle}(v,\mathrm{O}) = [\langle o\rangle \mapsto \langle o^v\rangle](\mathrm{O}).$$

The construction extends to the set of nodes $\mathrm{V}_{\mathcal{G}}$ of $\mathcal{G}$, whose instantiation is the union of the instantiation of each of its nodes:

$$\iota^{\langle o\rangle}(\mathrm{V}_{\mathcal{G}},\mathrm{O}) = \bigcup_{v\in\mathrm{V}} \iota^{\langle o\rangle}(v,\mathrm{O}).$$

Arcs The instantiation of an arc between two nodes adds links between darts images of the same initial dart via the two relabeling functions.

Figure 6.14: Instantiating an arc of a Jerboa graph scheme: (a) graph scheme with an arc between two nodes, (b) instantiation with the orbit graph of Figure 6.12a, and (c) instantiation with the orbit graph of Figure 6.12d.

The Jerboa graph scheme on the orbit $\langle 0,2\rangle$ depicted in Figure 6.14a is a graph with two nodes linked with a 0-arc. It corresponds to the graph of Figure 6.13a with the addition of the 0-arc. The instantiation of the nodes has already been discussed. The remaining task is the instantiation of the 0-arc $n0 \multimap n1$. The instantiation of $n0 \multimap n1$ adds links between copies of each dart from the initial orbit graph that correspond to either $n0$ or $n1$.

In the case of an initial free edge, node $n0$ yields two darts, $a0$ and $b0$, which are respective copies of the darts a and b from the initial orbit graph (see Figure 6.12a). Likewise, node $n1$ yields darts $a1$ and $b1$, which are the copies of darts a and b. Thus, we add the arcs $a0 \multimap a1$ and $b0 \multimap b1$.

We derive a similar construction in the case of an initial sewn edge. The orbit graph (see Figure 6.12d) contains four darts, meaning that nodes $n0$ and $n1$ instantiate into four darts each. The

instantiation of $n0 \multimap n1$ creates four links $a0 \multimap a1$, $b0 \multimap b1$, $c0 \multimap c1$ and $d0 \multimap d1$.

Definition 89 (Instantiation of the arcs)**.** *For s a node of $\mathscr{G}$ and u a node of* O*, we write (u,s) for the image of u in $\iota^{\langle o \rangle}(s,\mathrm{O})$.*

If the graph scheme $\mathscr{G}$ consists of two nodes s and t and an arc $s \overset{i}{\bullet\!\!-\!\!\bullet} t$, its instantiation with O *extends the instantiation of its nodes to link copies of the same node from* O*:*

$$\iota^{\langle o \rangle}(\mathscr{G},\mathrm{O}) = \underbrace{\iota^{\langle o \rangle}(s,\mathrm{O})}_{\text{node } s} \cup \underbrace{\iota^{\langle o \rangle}(t,\mathrm{O})}_{\text{node } t} \cup \underbrace{\bigcup_{u \in \mathrm{O}} (u,s) \overset{i}{\bullet\!\!-\!\!\bullet} (u,t)}_{\text{arc } s \overset{i}{\bullet\!\!-\!\!\bullet} t}$$

We write $\iota^{\langle o \rangle}(s \overset{i}{\bullet\!\!-\!\!\bullet} t,\mathrm{O})$ for $\bigcup_{u \in \mathrm{O}} (u,s) \overset{i}{\bullet\!\!-\!\!\bullet} (u,t)$.

If the graph scheme $\mathscr{G}$ is of the form (V,E)*, its instantiation with* O *extends $\iota^{\langle o \rangle}(\mathrm{V},\mathrm{O})$ to link copies according to all the arcs of* E*:*

$$\iota^{\langle o \rangle}(\mathscr{G},\mathrm{O}) = \underbrace{\iota^{\langle o \rangle}(\mathrm{V},\mathrm{O})}_{\text{nodes}} \cup \underbrace{\bigcup_{(v \overset{i}{\bullet\!\!-\!\!\bullet} v') \in \mathrm{E}} \iota^{\langle o \rangle}(v \overset{i}{\bullet\!\!-\!\!\bullet} v',\mathrm{O})}_{\text{arcs}}$$

To summarize, the instantiation of a Jerboa graph scheme intuitively corresponds to the following:

1. unifying the application of the relabeling functions (encoded by the orbit types on the node)
2. adding a link between the darts image of a node whenever there is an arc in the Jerboa graph scheme.

Jerboa rule scheme The instantiation of a Jerboa rule scheme $\mathscr{L} \overset{\langle o \rangle}{\longrightarrow} \mathscr{R}$ is defined as the instantiation of both $\mathscr{L}$ and $\mathscr{R}$ with the same orbit O of type $\langle o \rangle$. The resulting instantiations directly yield the graph transformation rule $\iota^{\langle o \rangle}(\mathscr{L},\mathrm{O}) \rightarrowtail \iota^{\langle o \rangle}(\mathscr{R},\mathrm{O})$.

Therefore, we can finally explain the reconstruction of the two rules of Figures 6.10a and 6.10b presented at the beginning of the section.

We consider the Jerboa rule scheme of Figure 6.11b. The left-hand and the right-hand sides are instantiated separately but with the same orbit graph. The instantiation of the right-hand side has already been discussed. The left-hand side consists of a single node $n0$ without any arc. Therefore, its instantiation is entirely described by the relabeling function deduced from its orbit type. In this case, the relabeling function is $\langle 0,2 \rangle \mapsto \langle 0,2 \rangle$, i.e., the identity function. In other words, the instantiations of the graph scheme with the graphs of Figure 6.12a and 6.12d yield these exact graphs. The right pattern is the graph scheme of Figure 6.14a.

Therefore, the complete instantiations of the Jerboa rule scheme correspond to the following:

- The instantiation of the left pattern on the graph of Figure 6.12a is the graph of Figure 6.12a, isomorphic to the left-hand side of the rule of Figure 6.10a.
- The instantiation of the left pattern on the graph of Figure 6.12d is the graph of Figure 6.12d, isomorphic to the left-hand side of the rule of Figure 6.10b
- The instantiation of the right pattern on the graph of Figure 6.12a is the graph of Figure 6.14b, isomorphic to the right-hand side of the rule of Figure 6.10a.

- The instantiation of the left pattern on the graph of Figure 6.12d is the graph of Figure 6.14c, isomorphic to the right-hand side of the rule of Figure 6.10b

In practice, the orbit type $\langle o \rangle$ for the rule parameter is given through the hook. We choose a node with no removing symbol '_' in its orbit type and use it as a reference to construct all relabeling functions. Thus, the hook serves a double purpose. On top of specifying where the modeling operation occurs in the object, it is used to build the relabeling functions. Applying a Jerboa rule scheme to a Gmap starts with the selection of dart a. From this dart, Algorithm 4 builds the orbit $\langle o \rangle(a)$ where the orbit type $\langle o \rangle$ is the one carried by the hook. The Jerboa rule scheme is instantiated with the orbit $\langle o \rangle(a)$ via instantiation of both its left-hand side and right-hand side. The instantiation provides a graph transformation rule applied to the initial Gmap.

In [16], the dimensions in the orbit types decorating the nodes of a Jerboa graph scheme are called *implicit arcs* since they implicitly represent relabeled links. By contrast, the arcs between nodes of a Jerboa graph scheme are called *explicit arcs*.

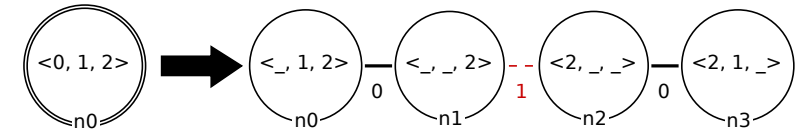
6.3.2 Rule schemes subsumes Jerboa rule schemes

As one can suspect from the explanation of the previous section, the instantiation of a discrete graph mimics the product on the set of nodes. Indeed, the copy of dart a obtained from node $n0$ is written $a0$, a short notation for $(a, n0)$.

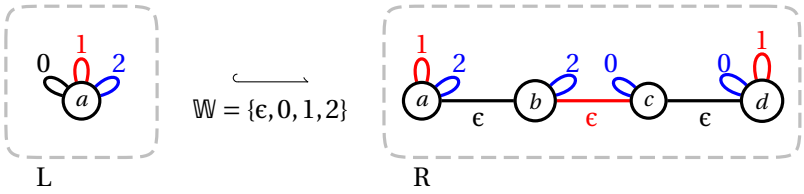
Let us compare the notion of Jerboa rule schemes and the notion of rule schemes as introduced in Chapter 4. Jerboa rule schemes are rules enriched with variables and instantiated with a set-based operation similar to a product. Orbits are defined as the transitive closure on a set of dimensions given a starting node, which corresponds to pattern graphs with one-letter words. The application of a Jerboa rule scheme substitutes the nodes with a copy of the orbit graph. Then, the relabeling functions change the labels of the links. Arcs in the rule are duplicated and link darts which are copies of the same initial dart in the orbit graph. Therefore, Jerboa rule schemes can be simulated with the framework of Chapter 4. The nodes of a Jerboa graph scheme correspond to the nodes of the graph scheme. Any i-arc between nodes of a Jerboa graph scheme is replaced by an (ϵ, i)-arc in the graph scheme. A relabeling of a dimension i into a dimension j deduced from the orbit type decorating a node in a Jerboa graph scheme yields a (i, j)-loop in the graph scheme.

The rule scheme for the quad subdivision of a 2-Gmap (already discussed in Chapter 4) is given again in Figure 6.15b. The rule scheme belongs to the category $(\{0,1,2\},\{0,1,2\})$-**Graph**. Recall that for such rule schemes, the color of the arc indicates the $\mathbb{D}$-part of the label, while the written word indicates its $\mathbb{W}$-part. For instance, the blue loop on node c, where the written label is 0, is a $(0,2)$-loop. Similarly, the red arc between nodes b and c is an $(\epsilon,1)$-arc. The counterpart Jerboa rule scheme on $\langle 0,1,2 \rangle$ is depicted in Figure 6.15a. The labels are explicitly written in such a representation, and the colors are there to ease the visualization. The double line around $n0$ is Jerboa's notation to specify that $n0$ is the hook. Both representations have the same number of nodes and arcs if we count the implicit arcs, i.e., the dimensions in the node decorations. A complete comparison of the two representations is given in Table 6.15c.

In the first part of this dissertation, we argued that rule schemes are more expressive than Jerboa rule schemes since they can be used for Omaps. Now that we have provided more details about Jerboa rule schemes and how to simulate them as rule schemes let us show a modeling operation on Gmaps that can be written as a rule scheme but not as a Jerboa rule scheme. We



(a)



(b)

Jerboa rule scheme (Fig. (a))	Rule scheme (Fig. (b))
Node	Node
$n0$	a
$n1$	b
$n2$	c
$n3$	d
Arc	Arc
$n0 \overset{0}{\text{—}} n1$	$a \overset{(\epsilon,0)}{\text{—}} b$
$n1 \overset{1}{\text{—}} n2$	$b \overset{(\epsilon,1)}{\text{—}} c$
$n2 \overset{0}{\text{—}} n3$	$c \overset{(\epsilon,0)}{\text{—}} d$
Orbit type on the node	Loop(s) on the node
$\langle 0,1,2\rangle$ on $n0$ (left)	$(0,0)$, $(1,1)$, and $(2,2)$ on a (left)
$\langle \_,1,2\rangle$ on $n0$ (right)	$(1,1)$ and $(2,2)$ on a (right)
$\langle \_,\_,2\rangle$ on $n1$	$(2,2)$ on b
$\langle 2,\_,\_\rangle$ on $n2$	$(0,2)$ on c
$\langle 2,1,\_\rangle$ on $n3$	$(0,2)$ and $(1,1)$ on d

(c)

Figure 6.15: Comparison between the rule scheme and the Jerboa rule scheme encoding the quad subdivision of a surface: (a) the Jerboa rule scheme, (b) the rule scheme, and (c) the complete comparison.

consider a 2D operation subdividing a face of even arity, i.e., with an even number of edges, into quads. Instead of adding a vertex at the middle of each initial edge, we propose to connect the vertex added at the center of the face to every other initial vertex. Results of this operation are respectively illustrated with a square in Figure 6.16a and a hexagon in Figure 6.16b. We can express this operation with the rule scheme of Figure 6.16c. This is a $(\{\epsilon, 1, 010\} \times \{0, 1, 2\})$ rule scheme. The operation can not be translated into a Jerboa rule scheme. Indeed, the rule scheme uses words with more than one letter. Besides, the hook (node a, as it is the only node in the left-hand side) does not have a loop for each word in $\{\epsilon, 1, 010\}$, meaning that the whole face is not matched. Intuitively, the only possibility to modify a face regardless of its topology is to use the orbit $\langle 0, 1\rangle$, which then prohibits providing different treatments to the individual darts of the face. Note that this operation is still expressible as a Jerboa rule scheme, but one such rule needs to be specified for each arity.

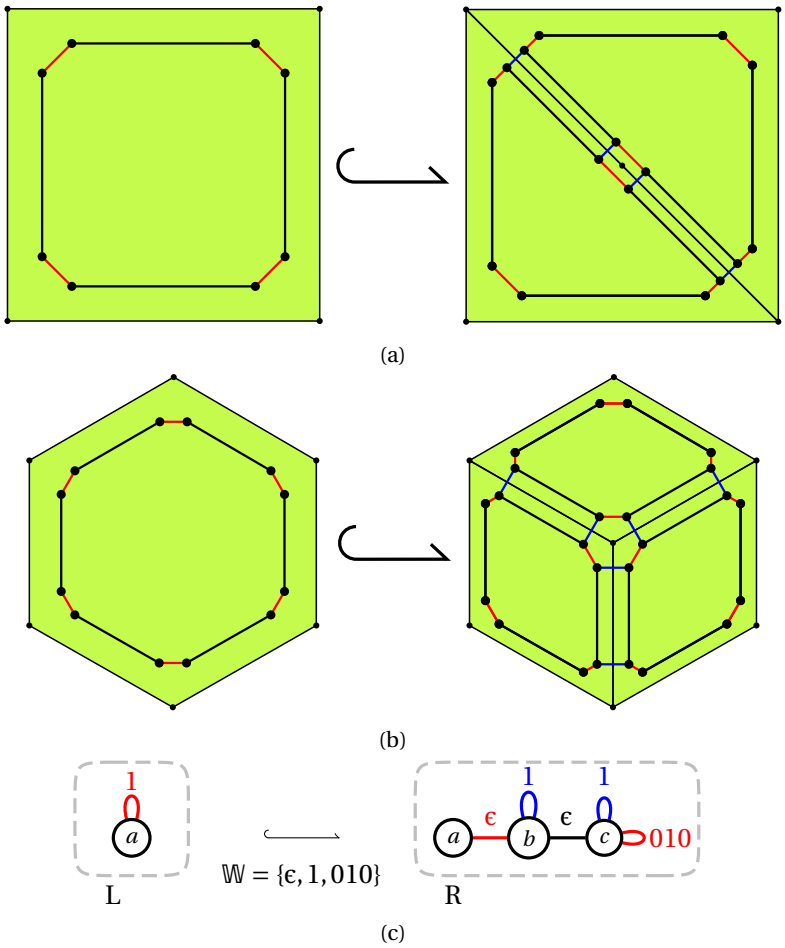
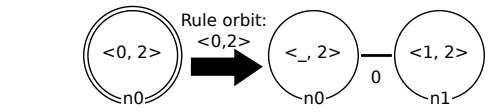


Figure 6.16: Subdividing an even face into quads by adding edges to every other vertex: (a) illustration on a square, (b) illustration on a hexagon, (c) the corresponding rule scheme, which cannot be expressed with a Jerboa rule scheme.

6.3.3 Embedded Jerboa rule schemes

In the formal part of the dissertation, we covered the manipulation of embedding values via attributed rules. In Jerboa rule schemes, a node encodes the topological transformation for several darts. The node also encodes the geometric transformation of these same darts. The embedding expressions of Chapter 5 can be written as-is on Jerboa rule schemes, barring the use of node identifiers instead of dart identifiers. After the instantiation of a Jerboa rule scheme, the node identifiers are substituted with the dart identifiers (which further justifies the construction of Section 5.5). In other words, the embedding expression of a node is duplicated on each of its instantiated darts, but the dart identifier replaces the node identifier. Then, the computation is realized as explained in Chapter 5. Recall from Chapter 5 that all darts in the same embedding orbit should have the same embedding value.

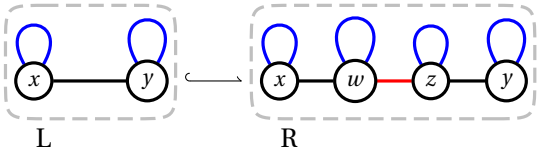
Let us consider the vertex insertion operation again, in its version folded with both 0 and 2, i.e., the Jerboa rule scheme of Figure 6.17a. The right-hand side node $n0$ is a preserved node, meaning that all its instantiated darts will be preserved darts in the instantiated rule. Therefore, if no new embedding value is assigned to these darts, they will keep their value from before the operation. In other words, node $n0$ does not need any embedding expression. On the contrary, node $n1$ is an added node, which instantiates into added darts. We give an embedding expression to $n1$. Assum-



(a)

Node	col	pos
$n1$	$n0.col$	$mix(\langle 0\rangle_{pos}(n0))$

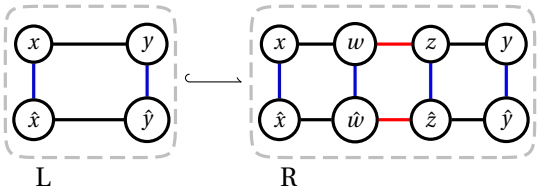
(b)



(c)

Dart	col	pos
w	$x.col$	$mix(\langle 0\rangle_{pos}(x))$
z	$y.col$	$mix(\langle 0\rangle_{pos}(y))$

(d)



(e)

Dart	col	pos
w	$x.col$	$mix(\langle 0\rangle_{pos}(x))$
z	$y.col$	$mix(\langle 0\rangle_{pos}(y))$
$\hat{w}$	$\hat{x}.col$	$mix(\langle 0\rangle_{pos}(\hat{x}))$
$\hat{z}$	$\hat{y}.col$	$mix(\langle 0\rangle_{pos}(\hat{y}))$

(f)

Figure 6.17: Instantiation of embedded Jerboa rule schemes: (a) Jerboa rule schemes for the vertex insertion, (b) embedding expressions associated with the nodes of the rule from Figure (a), (c) rule for the vertex insertion in a free edge, (d) embedding expressions associated with the darts of the rule from Figure (c), (e) rule for the vertex insertion in a sewn edge, and (f) embedding expressions associated with the darts of the rule from Figure (e).

ing a representation with colors and positions, we can give the expressions of Table 6.17b. For the color embedding, we add the same color as $n0$, i.e., specify the expressions $n1.col = n0.col$. For the position embedding, we insert the vertex at the middle of the initial edge via the expression $mix(\langle 0\rangle_{pos}(n0))$. The topological instantiations with a free, resp. sewn edge, have been given in Figures 6.10a and 6.10b and are recalled in Figures 6.17c and 6.17e.

In the case of the initial free edge, the embedding expressions on the instantiated rule are given in Table 6.17d. Only w and z have expressions as they are instantiated from $n1$. For both embeddings, the two darts have different embedding expressions. However, x and y are 0-linked in the left-hand side, and the color embedding is carried by the orbit type $\langle 0,1\rangle$. Therefore, x and y have the same color value, and we get $w.col = x.col = y.col = z.col$. Since x and y are 0-linked, the expressions $mix(\langle 0\rangle_{pos}(x))$ and $mix(\langle 0\rangle_{pos}(y))$ will yield $\frac{1}{2}x.pos + \frac{1}{2}y.pos$. In other words, the expressions always result in equal values for w and z.

The case of the initial sewn edge is given in Table 6.17f. In this case, x and y belong to the same $\langle 0,1\rangle$-orbit in the left-hand side, while $\hat{x}$ and $\hat{y}$ belong to another $\langle 0,1\rangle$-orbit. Therefore, we obtain $w.col = x.col = y.col = z.col$ and $\hat{w}.col = \hat{x}.col = \hat{y}.col = \hat{z}.col$, but the two values can be different. Different values are acceptable as they do not break the geometric consistency for the color embedding. For the position values, we obtain $\frac{1}{2}x.pos + \frac{1}{2}y.pos$ for darts w and z, and $\frac{1}{2}\hat{x}.pos + \frac{1}{2}\hat{y}.pos$ for darts $\hat{w}$ and $\hat{z}$. Exploiting that $x.pos = \hat{x}.pos$ and $y.pos = \hat{y}.pos$, since the darts respectively belong to the same $\langle 1,2\rangle$-orbit, we obtain that values for w, $\hat{w}$, z, and $\hat{z}$ are equal.

We discussed and proved conditions on DPO rules to ensure that the application of a rule does not break the geometric consistency. Conditions on Jerboa rule schemes have been studied in [14] to ensure the conditions on the instantiated rules. Essentially, the term substitution realized on the attributes should lead to equal values for darts within the same embedding orbit. Note that

even though darts may be instantiated from the same node, the substitution mechanism perform the computation on a dart basis. In particular, an expression may yield different values for two darts instantiated from the same node and belonging to the same embedding orbit. Embedding expressions are considered up to an equivalence relation based on the orbit type to avoid such inconsistencies. Intuitively, we obtain another syntactic condition that guarantees the preservation of the embedding consistency. In practice, a rule might not satisfy the condition while still being consistent. Indeed, user-defined embedding computations may have further properties, such as commutativity or associativity, which are not considered when computing the equivalence of terms.

In practice, geometric consistency can also be exploited on two levels. The embedding propagation (see Chapter 5) adds the computed value to all darts added by the topological extension. Therefore, the actual computation is realized only once and then propagated, which prevents duplicated computations [16]. From this practical exploit, we derive a second simplification. This simplification occurs at the Jerboa scheme rule level. Because we know that only one value needs to be computed per orbit, we can specify only one computation per 'orbit' of explicit dimensions in the Jerboa rule scheme. Although orbits are not properly defined on Jerboa rule schemes, the idea is to reason on the topological counterpart of a Jerboa graph scheme where the decoration on nodes, i.e., the orbit types, are not considered. For instance, in the computation of the quad subdivision depicted in Figure 6.15a, all nodes of the right-hand side belong to the same $\langle 0,1\rangle$-orbit. Thus, only one embedding expression has to be specified for the color embedding. Besides, if we do not want to modify the color, we can keep the color of the initial faces by not specifying any embedding expression. In such a case, the values of the darts instantiated from $n0$ will be propagated.

As a final remark on the question of embedded Jerboa rule schemes, the script language developed by Valentin Gauthier during his Ph.D. thesis [74] allows writing embedding expressions in an imperative style.

6.3.4 Jerboa's rule application engine

In Chapter 4 and in Section 6.3, we described the application of a scheme rule as three intermediate steps. First, the rule scheme is instantiated to obtain a rule in the framework of DPO rewriting. This step embodies the topological modification of the modeling operation. Secondly, the embedding expressions, i.e., the terms that describe the geometric modifications, are substituted, and the values after the transformation are computed. Lastly, the instantiated rule with computed values is applied, which produces the transformed Gmap and, thus, the transformed object.

In practice, Jerboa's rule engine performs the instantiation, the substitution, and the application simultanously [12]. The application of an operation needs to consider both topological and geometric modifications. The topological part relies on computing a table that encodes the neighboring relations of the modified darts before and after the transformation. Once the table has been filled, the geometric computations are performed and stored in a separate buffer. Performing the computations outside the modified Gmap ensures no concurrent read or write access to the embedding values. Once the embedding computations are performed, the modified darts are aggregated with the Gmap by adding the missing links. In practice, links are overwritten rather than deleted and added again since the rules' conditions ensure that the obtained graph is a well-

formed Gmap. Finally, the computed embedding values are propagated based on the orbit type of each embedding. The embedding propagation is done after the aggregation of the topological modification to avoid dealing with the topological extension. Note that for the parallel case, the application of an operation still relies on a matrix construction which is slightly different as discussed in [26].

6.4 Applications done with Jerboa

In this section, we present some applications done using the Jerboa framework. The primary motivation for the rule-based approach defined in the first part of this dissertation is to develop the Jerboa platform to support the design of modelers dedicated to different application areas. In [3], we described an application for computing exploded views of objects to help visualize their inner structure. The result, called JerboaEclatement,[5] contains a unique rule with various parameters, used to split topological cells. Examples of results are given in Figure 6.18.

(a) (b) (c)

Figure 6.18: Exploded views of a nightstand: (a) original, (a) explosion per volumes, and (a) explosion per faces.

More complete applications have also been developed for academic and industrial projects. Jerboa has been successively used for several modelers:

- JerboaArchi is a modeler dedicated to architecture (Figure 6.19a), providing basic operations for elevating/extruding a 2D map (done by an architect) into a 3D map, for example, operations to add doors or windows. This modeler has been used to experiment with the reevaluation of rule sequences [30]. It allows the recording of an interactive construction to be reevaluated with new geometric parameters, creating a new model.

- Jermination follows L-system mechanisms [21] appreciated by botanists, who usually write rules reflecting the elementary steps of plant growth. Jermination implements similar rules and allows the display of the growth stages of a plant (Figure 6.19b).

- Jeolog is an industrial modeler dedicated to geology (Figure 6.19c). Here also, the flexibility of embeddings allows a double representation of geometric points (one at sedimentation times and another at present times) [74]. This feature eases the comprehension of Earth

[5] Available on Jerboa's website: `http://xlim-sic.labo.univ-poitiers.fr/jerboa/doc/jerboaeclatement/`.

layers and fault displacements. Topological cells store data, which simplifies the design of complex operations.

- Japhy stands for Jerboa animation-based physics and provides a library for multi-physics simulations (Figure 6.19d). Currently, it supports some physics models such as mass-spring, finite element method, or mass-tensors among several meshes (triangle, quad, tetrahedron, hexagon). Rules help developers correctly design the location of force interactions [17], while runtime verification helps them write correct computations when designing new forces.

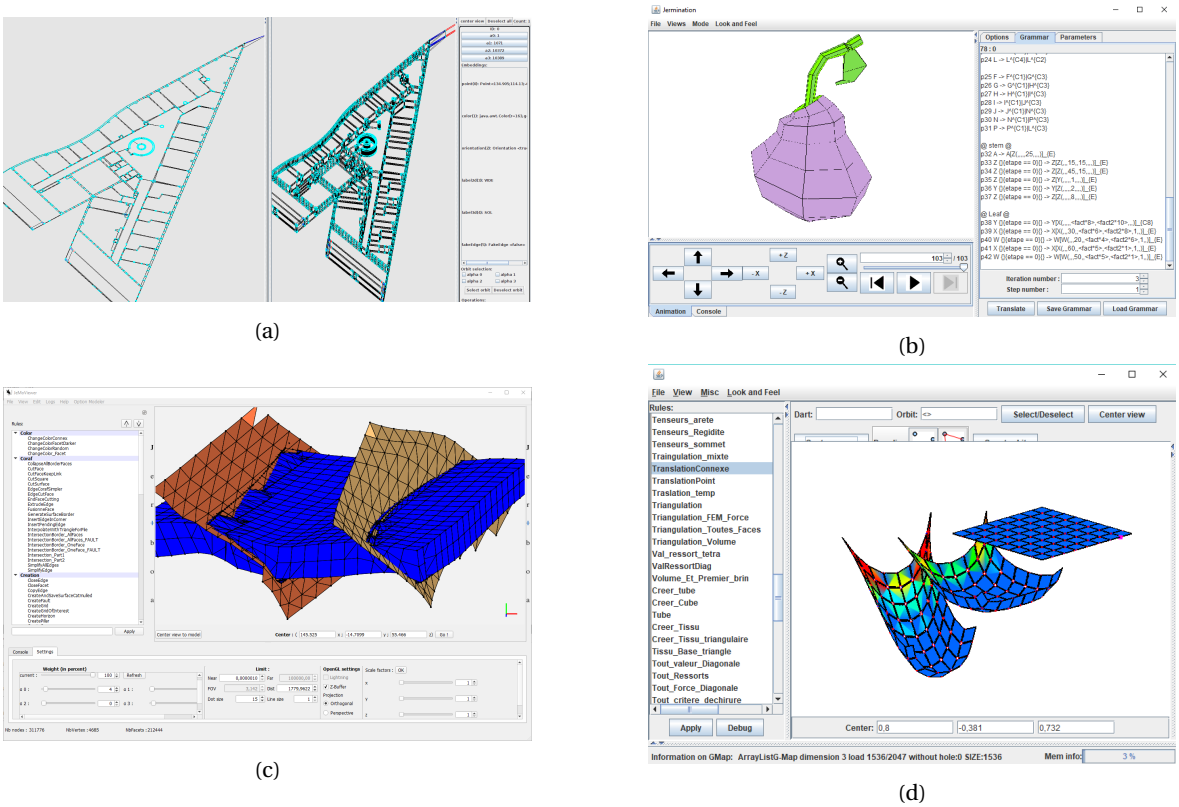


Figure 6.19: Various modelers designed with Jerboa: (a) JerboaArchi, (b) Jermination, (c) Jeolog, and (d) Japhy.

Figure 6.19 gives a brief overview of these modelers designed using Jerboa. These applications show interest in the fast prototyping of modelers for dedicated domains, which, in return, motivates us to ease the design and verification of modeling operations.

6.5 On the difficulty of designing modeling operations with Jerboa

In his Ph.D. thesis [74], Valentin Gauthier designed *scripts* as an extension of Jerboa rule schemes. Informally, Jerboa rule schemes were turned into functions in a programming language sense. Therefore, scripts allow for the chaining of rules within a dedicated language, simplifying the design of more complex operations. The language also allowed for an extension of the embedding expressions from a functional approach to an imperative approach. Although the designed language is syntactically similar to most programming languages, the modeler designer needs to learn the scripting language on top of learning how to design Jerboa rule schemes.

Valentin Gauthier also showed that composing Jerboa rule schemes may speed up computation times, especially when computing iterations of subdivision schemes. In such cases, the size of the Jerboa rule scheme exponentially increases with the number of iterations simulated, which renders them hard to read, and, usually, if not for the composition construction, even harder to write. As implemented, the composition of Jerboa rule schemes offers several limitations. The composition mechanism only covers Jerboa rule schemes where the left-hand side contains a unique node. Some compositions lead to topologically invalid Jerboa rule schemes. Such cases occur when the second Jerboa rule scheme tries to access elements that are not in the occurrence of the match from the first Jerboa rule scheme. Lastly, the composition only covers the topological aspects, and the embedding expressions still need to be added manually. Despite these limitations, this composition mechanism offered the first solution to deal with harder-to-obtain geometric modeling operations.

Rather than extending Jerboa's language again with the hope of simplifying the design of even more complex operations, we propose a somewhat reverse approach. We consider only operations expressible as Jerboa rule schemes but discharge the designer of the cumbersome task of writing them. Indeed, rules offer a specific framework that can guide us to retrieve the operation. The following two chapters are dedicated to a method for inferring a Jerboa rule scheme from a representative example. With the help of additional information and some hypotheses on the embedding expression, our proposed method allows retrieving both the topological and geometric components of the Jerboa rule scheme.

Summary of the chapter's contributions

The purpose of this chapter was twofold. First, we illustrated how to use the framework presented in the previous chapters. We hope this chapter has clarified some categorical notions and constructions considered with the eyes of a developer. In particular, we hope that the construction and manipulation of Jerboa rule schemes shed light on rule schemes. In the following chapters, we will present the inference of geometric modeling operations as Jerboa rule schemes. However, we will no longer distinguish between the two notions and call a 'rule scheme' a Jerboa rule scheme. As a second purpose, we have presented the tool that we will use in the following chapters, Jerboa, and some of its algorithms. We will reuse the construction of the instantiation mechanism via relabeling functions in Chapter 7. We will also exploit, in Chapter 8, the geometric considerations discussed here. Most notably, we will rely on how embedding expressions are carried out from rule schemes to instantiated rules and why we do not necessarily need to specify embedding expressions on all nodes.

Chapter 7

Inference of topological rule schemes

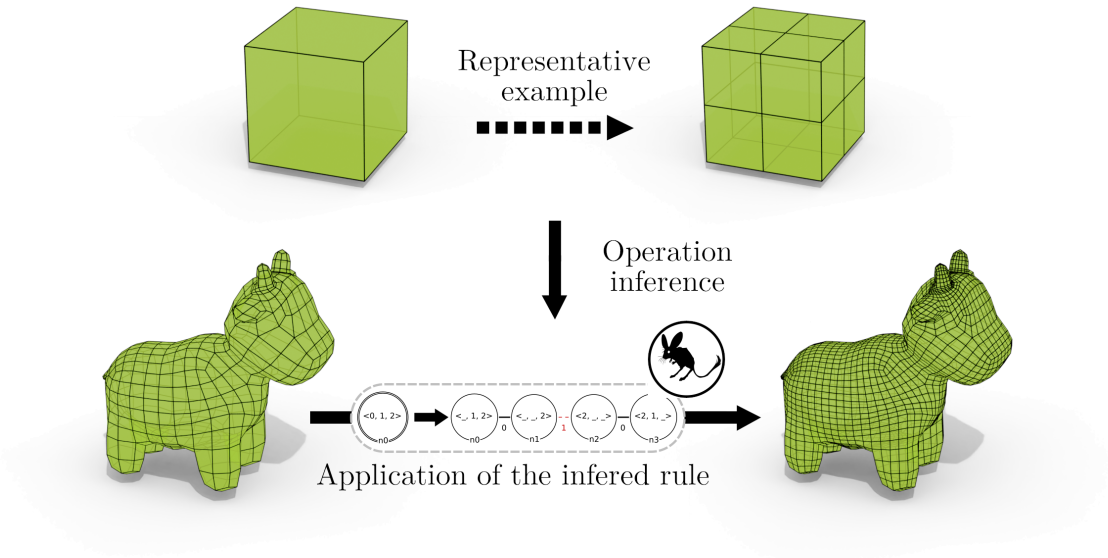


Figure 7.1: We aim at inferring a generic rule from a representative example.

Personal note on the chapter

Trends in research being what they are, I started my Ph.D. thesis with the idea of using machine learning to reconstruct modeling operations, borrowing inspiration from techniques used for inference from sketches and learning on graphs, such as approaches using graph neural networks [159]. After working on the theoretical formalization presented in the first part of this dissertation, I realized that such tools might not be needed. Granted that we aimed to reconstruct rule schemes as expressed in Jerboa, we could use our knowledge of the instantiation process to reverse it. We had some meetings where we discussed how to do it practically, but I was not fully able to convince everyone that properly folding a Gmap should yield the desired results. The key idea was to fold the graph locally and spread the folding by traversing the structure rather than approaching the problem globally. Ultimately, I showed a proof of concept of what became the topological folding algorithm, managing to infer the quad subdivision.

Contents

In this chapter, we present a method to infer the topological part of a modeling operation from a representative example. The motivation is twofold. The first motivation comes from the desire to solve a recurrent issue in geometric modeling, where the design of correct operations is known to be time-consuming and challenging. However, these operations are intuitively understood via simple drawings of a representative object before and after modification. In the first part of this dissertation, we saw that rule schemes alleviate the difficulty of designing correct operations via a mathematically sound formalization of modeling operations as rules. However, their authoring remains difficult. Designing rule schemes requires learning Jerboa's domain-specific language and a good knowledge of generalized maps, leading to our second motivation. Jerboa may appear as a difficult tool to use at first sight, especially because of the language used to design modeling operations. To tear down this overemphasized barrier, we propose to infer rule schemes automatically based on an instance of the operation, hence providing a solution that simplifies the design of new operations and hides the technical elements.

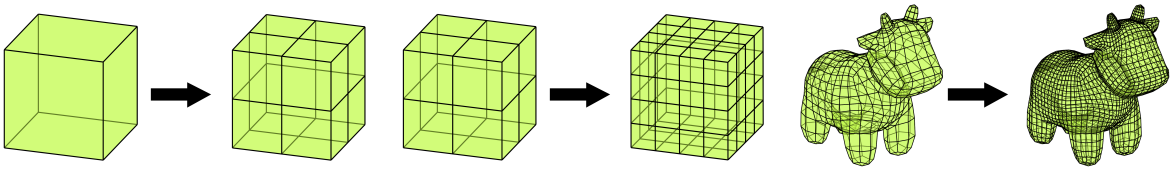
We essentially used two inspirations for the key construction of the inference mechanism. The first one comes from the categorical formalization of modeling operations as rule schemes. The crucial part of the instantiation mechanism is the product between the pattern graph and the graph scheme. Therefore, the inference of a rule scheme (or a graph scheme) can be narrowed down to reversing this product, i.e., performing a graph 'quotient.' Note that apart from a short digression, we will not present the inference of operations in a categorical formalism. Nonetheless, I would like to emphasize that the elaboration of the topological folding algorithm deeply roots itself in the intuition that we are, in a way, reversing a product of graphs. The second intuition comes from the instantiation mechanism from relabeling functions. When we display the instantiated rule (or graph), we can identify recurrent parts in the graph delimited by symmetry axes. These recurrent patterns can be described by considering the instantiation mechanism from the perspective of a dart in the initial orbit rather than the graph scheme's nodes. A more careful examination reveals that these patterns encode the implicit arcs when a link crosses one symmetry axis and the explicit arcs when a link stays within the pattern.

These two intuitions will be clarified in Section 7.1, along with comments on approaches found in the literature to solve similar problems and a discussion about the more global workflow for the inference of modeling operations. The main contribution of this chapter is the topological folding algorithm presented in Section 7.2. While providing the algorithm, we will detail both successful and infructuous cases before generalizing from graph schemes to rule schemes. The ambition to infer geometric modeling operations led us to develop a dedicated tool called Jerboa Studio. The presentation of Jerboa Studio in Section 7.3 is accompanied by a discussion about practical elements necessary for the implementation of the topological folding algorithm. Section 7.4 is dedicated to the presentation of some results obtained with our algorithm, namely an application in geology and an application for subdivision schemes. Lastly, Section 7.5 sketches an analysis of the algorithm detailed in Appendix C. Note that this chapter is exclusively of topological content. We will explain how to retrieve the geometric computations in the following chapter.

7.1 Motivation

In interactive modeling, the possibility to effortlessly create dedicated operations is a long-craved ambition. These operations aim at simplifying the production of domain-specific objects. Geometric modelers [80, 168, 33] usually enable the user to hand-code new operations through an API, adapting a generic tool into a dedicated one. Our ambition is to deduce the general operation from a single representative instance. Indeed, domain experts usually experiment on simple instances when the target object is complex, to the point that they can often characterize an operation using a well-chosen use case. Besides, inferring operations from an instance reduces the cumbrous nature of implementing new operations, coping with the unfamiliarity of domain experts with the tool's implementation. We aspire to exploit the intuition that experts can provide to infer operation in the specific case of topological modifications on meshes.

We wish to provide a tool that can infer the topological part of an operation from a representative example. The operation should be applicable to similar objects. The user specifies an initial object A, modifies it, and provides the final object B. We offer to deduce the operation that directly transforms A into B. For example, Figure 7.2a illustrates the quad subdivision on a cube. In this case, object A is the cube before the subdivision, while object B is the cube after the subdivision. Since we want an operation applicable in a broader context than simply for object A, we infer operations generic up to a user-specified orbit, i.e., a rule scheme. For instance, if we infer the quad subdivision generic up to a surface, we seek to obtain the rule scheme provided in Chapter 6. The inferred operation should be applicable again to the resulting object (see Figure 7.2b) or to different objects, such as the quad mesh of a cow depicted in Figure 7.2c.



(a) (b) (c)

Figure 7.2: Quad subdivision: application to a cube (a), iterated application to the cube (b), and application to a quadmesh (c).

7.1.1 Related works

The inference of operations has already been studied within both computer graphics and graph transformation communities. Before providing more details about our approach, we review some other methods within both these fields, clarifying why these approaches do not offer valid solutions for our concerns.

Inverse procedural modeling Procedural modeling (described in Chapter 1) refers to techniques used in computer graphics to derive a model from a ruleset. Since these techniques could not guarantee an output faithful to the designer's idea, they were extended to inverse procedural modeling. These new approaches try to discover the correct parameterized rules and values thanks to machine learning. Inverse procedural modeling techniques have proven successful in most of the domains where procedural modeling was used, namely for trees [170], building facades [185],

weather simulation on urban models [73], virtual worlds [65], and texture modeling [103]. In such approaches, the set of possible rules describes a specific domain, and the operation inference is tailored to this domain. Conversely, the approach that we will present in this dissertation builds topological operations that allow editing objects regardless of the application field. Indeed, our rule-based approach discovers the correct rule parameterized by topological information and not by domain-specific values.

L-systems In [169], the authors use a clustering approach to construct the rules and parameters of a parametric context-free L-system, given a vectorial 2D image. More recently, [83] has extended the work of [169] with deep learning techniques for detecting elements and a derivation tree for the retrieval of the rules. In these works, the emphasis is put on the generation of scenes while we focus on the inference of operations. L-systems have been used to represent the refinement operation for subdivision curves [149], where the authors show how to infer an L-system from the subdivision matrix. However, L-systems are essentially geometric interpretations of words, thus inherently equivalent to string rewriting. Therefore, they are ill-suited for working on surfaces and volumes. Graph rewriting extends string rewriting, finding an occurrence of a graph and replacing it with another graph. In a sense, Gmap rewriting represents a more general approach than L-systems. Therefore, the ambition of the present dissertation can be understood as a generalization of [158] from L-systems to graph rewriting.

Reevaluation To avoid cumbersome implementation, some modelers support the definition of modeling operations by recording a sequence of already existing operations. The reevaluation [114, 68] of the sequence provides a solution to apply this new operation via a specific naming of the modified entities [30]. Thus, the reevaluation method allows for modifying similar objects, i.e., objects with the same entity naming. However, the constructed operation is often not an optimized solution to define the modification because every step is reproduced faithfully.

Neural networks and geometrical approaches Recently, [123] offered an automatic generation of geometric operations. This approach takes the Loop subdivision scheme [124] as the atomic operation for refinement. A neural network is then used to learn the geometric values for the subdivision. Therefore, the approach produces the result from an initial object through the direct computation of the targeted geometry. This construction is dual to ours as we focus only on the topology. Nonetheless, our generalization power is broader as the inferred operations do not assume a fixed topology. For instance, we can reconstruct any subdivision scheme.

Inference in graph transformation Our approach exploits a domain-specific language within graph transformations for topology-based geometric modeling. We already discussed some approaches for the inference of graph transformations in the introduction, emphasizing that our method can be compared to the one from [99, Chap. 8], which retrieves visual contracts. Indeed, our ambition is to reconstruct both structural and attribute modifications. The structural part describes topological changes, which is the topic of this chapter, while the attribute one describes embedding modifications discussed later in Chapter 8. Apart from the distinct application domain, our inference mechanism differs from visual contracts by the kind of input used to reconstruct the rules. We infer from an instance of an operation rather than from the traces of pro-

grams, meaning that we have graph structures readily available. We also require more information from the user, such as a parameterization of the rule and some mapping information. Our approach also shares similarities with [125], which reconstructs a graphical modeling environment for domain-specific language using yED.

Inference of modeling sequences in constructive solid geometry In [187], the authors infer the sequence of modeling operations as a sequence of sketches, extrusions, and boolean operations. It extends previous works such as [162], [49] or [109] using constructive solid geometry (CSG). These works retrieve a CSG tree to obtain a specific object. The deduced tree yields an exact object construction but does not endow a definition of operations. One could easily use these techniques to build the first iteration of a subdivision scheme. However, the obtained tree would not give a solution to build the successive iterations of the subdivision scheme.

7.1.2 General workflow

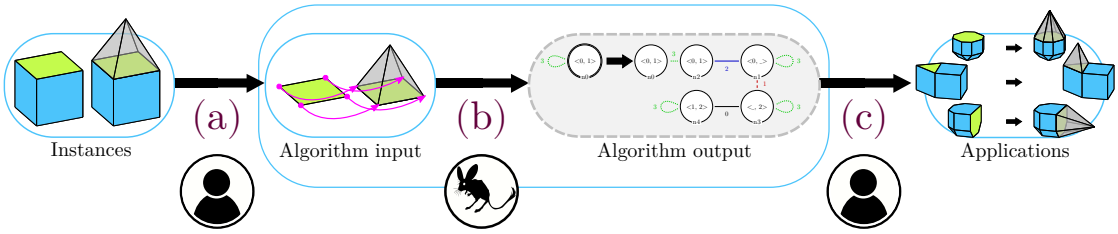


Figure 7.3: General workflow for the inference of a modeling operation: the user provides two instances adequately representing the modeling operation; (a) they build a partial mapping describing the preserved elements, (b) our topological folding algorithm reconstructs a rule scheme generalizing the topological part of the operation, (c) the user adds the missing embedding expressions before exporting the rule to code; from the exported rule, the user obtains an operation readily applicable to other objects.

Before providing our topological folding algorithm, we will discuss which steps are automated and which ones require user interaction. The general workflow is illustrated in Figure 7.3, where the Jerboa pictogram under (b) signifies that Jerboa automates the step, and the character pictogram under (a) and (c) means that user interaction is required. First, the user builds two instances of an object as a representative example of the operation before and after its application. The idea is to provide an example that adequately describes the operation. Secondly, the user specifies a mapping of the preserved elements. This step ensures that the deduced operation properly preserves unmodified elements within its scope (step (a) in Figure 7.3). Before running the topological folding algorithm, the user can specify an orbit type that should correspond to the rule's parameter, i.e., an orbit type for one of the hooks. Thirdly, the topological folding algorithm exploits the user-specified information to deduce a compatible operation (step (b) in Figure 7.3). If the topological folding algorithm provides an output, the user obtains a valid rule scheme describing the topological part of the operation. Note that inferred operations are equivalent to ones designed by the user and meet the same requirements for generation and use in the viewer. If the user wants to apply the inferred operation, they need to handle the geometric modifications that occur in the transformation. Since the inferred rule scheme is similar to a manually generated one, the user can edit the result of the algorithm to add the missing embedding expressions. The user

will be guided by the rule editor, which will specify the nodes that need an expression and whether or not the provided expressions are valid. The addition of the embedding expression corresponds to the last step before generating the rule (step (c) in Figure 7.3), which will be automated in the next chapter.

7.1.3 Intuition supporting the conception of the algorithm

Intuition from category theory

In Chapter 4, we presented the instantiation of a rule scheme through a functorial approach. From the choice of a pattern graph, we were able to build a rule acting on Gmaps from a rule scheme where arcs are labeled with pairs encoding the creation of arcs from a path label. Here, we seek to decompose a Gmap into a pattern graph and a graph scheme, such that the instantiation of the graph scheme on the pattern graph yields the initial Gmap. In practice, we do not use any pattern graph for the instantiation. Indeed, the pattern functor for a set of words $\mathbb{W}$ is first applied to the topological graph before choosing a strongly connected component in the resulting graph. This construction only depends on the choice of $\mathbb{W}$, which corresponds to the choice of the orbit type used to parameterize the operation. We can typically ask the user for this information, meaning that we can simulate some part of the instantiation mechanism, i.e., up to the application of the embedding functor. At this point, we have the topological graph, which belongs to the category $\mathbb{D}$-**Graph** and the embedded pattern graph, which belongs to the category $(\mathbb{W} \times \mathbb{D})$-**Graph**. The information that we are seeking is the missing graph scheme. Two steps would then be missing to end the instantiation. First, we would build the product between the graph scheme and the embedded[1] pattern graph. Secondly, we would apply the projecting functor to obtain the instantiated graph. The projecting functor is not an isomorphism of categories, as it intuitively deletes information. Otherwise, we could apply the inverse functor and build the graph scheme as the final pullback complement used in the Sesqui-Pushout approach to graph transformation [36], simulating the product as a pullback on the terminal object. Therefore, we need another solution to retrieve the $\mathbb{W}$ part of the arc labels. We propose to make 'educated guesses' from the fact that the graph onto which the projecting functor is applied is a product. Intuitively, we want to reverse the product on the topological graph while ensuring that correct labels can be reconstructed. We will exploit the construction of the categorical product, the topological constraints on Gmaps, and their counterpart conditions on rule schemes. We will see that the graph scheme can be unambiguously retrieved up to one choice. Loops on the graph scheme can sometimes be labeled either (ϵ, i) or (d, i). Such a case only happens from a lack of information in the instantiated graph, meaning that the example provided for the inference was not representative enough. A careful comparison of the presented algorithm with the constructions from Chapter 4 reveals that the central part of our inference mechanism revolves around reversing the product of graphs while recreating the missing $\mathbb{W}$ part of the arc labels: we intuitively perform some graph 'quotient.'

Intuition from the visual representation of the instantiation mechanism.

In Chapter 6, we presented the instantiation of a rules scheme via relabeling functions. When we display the instantiated rule (or graph), we can identify recurrent parts in the graph. We explain

[1] Here 'embedded' is to be understood in the sense of the embedding functor seen in Chapter 4 (see Definition 41).

this intuition with the help of Figures 7.4a and 7.4b, displaying a zoom on the surface undergoing a quad subdivision. Figure 7.4a displays the Gmap before the transformation, and Figure 7.4b after the transformation. The surfaces are also drawn underneath. If we want to infer an operation modifying entire surfaces, then the resulting rules scheme should have a hook decorated with an orbit type $\langle 0,1,2\rangle$. Besides, such a hook would be associated with all darts of the surface, meaning that the left-hand side of the rule could not have another node. Therefore, we know the motif to examine consists of a single dart. In Figure 7.4c, we zoomed on a part of the modified object and isolated each dart. Brown dashed lines split the darts between their 0-link, purple dashed lines between their 1-link, and pink dashed lines between their 2-link. These dashed lines delimit triangles bordered by one brown, one pink, and one purple side. The object is entirely described by a tiling made of these triangles. The content within one triangle is enough to reconstruct the entire surface, provided that we properly glue the triangles along sides of the same color. We now replicate these triangles on the Gmap after the transformation, as shown in Figure 7.4d. The triangulation is the same, but the content within each triangle has been modified. We can isolate a triangle to describe the modifications made (see Figures7.4e and7.4f).

- The dart has been replaced by four darts.
- The brown lines which were crossed by a 0-link are now crossed by two 2-links.
- The purple lines which were crossed by a 1-link are now crossed by two 1-links.
- The pink lines which were crossed by a 2-link are now crossed by two 2-links.

If we forget the inner content of one triangle and only focus on the links crossing each color, we can still reconstruct the complete triangulation by stitching the links. Therefore, we can distinguish the inner content, which provides the explicit arcs of the rules scheme, from the links crossing the lines, which describe the relabeling and, therefore, the implicit arcs. The process of inferring an operation can then be intuitively described as retrieving the atomic motif defining the operation.

On a side note, the description of modeling operations with recurrent motifs matches the description of folding and unfolding from the previous chapter. Therefore, we call *topological folding algorithm* our inference mechanism, which we will present in the next section.

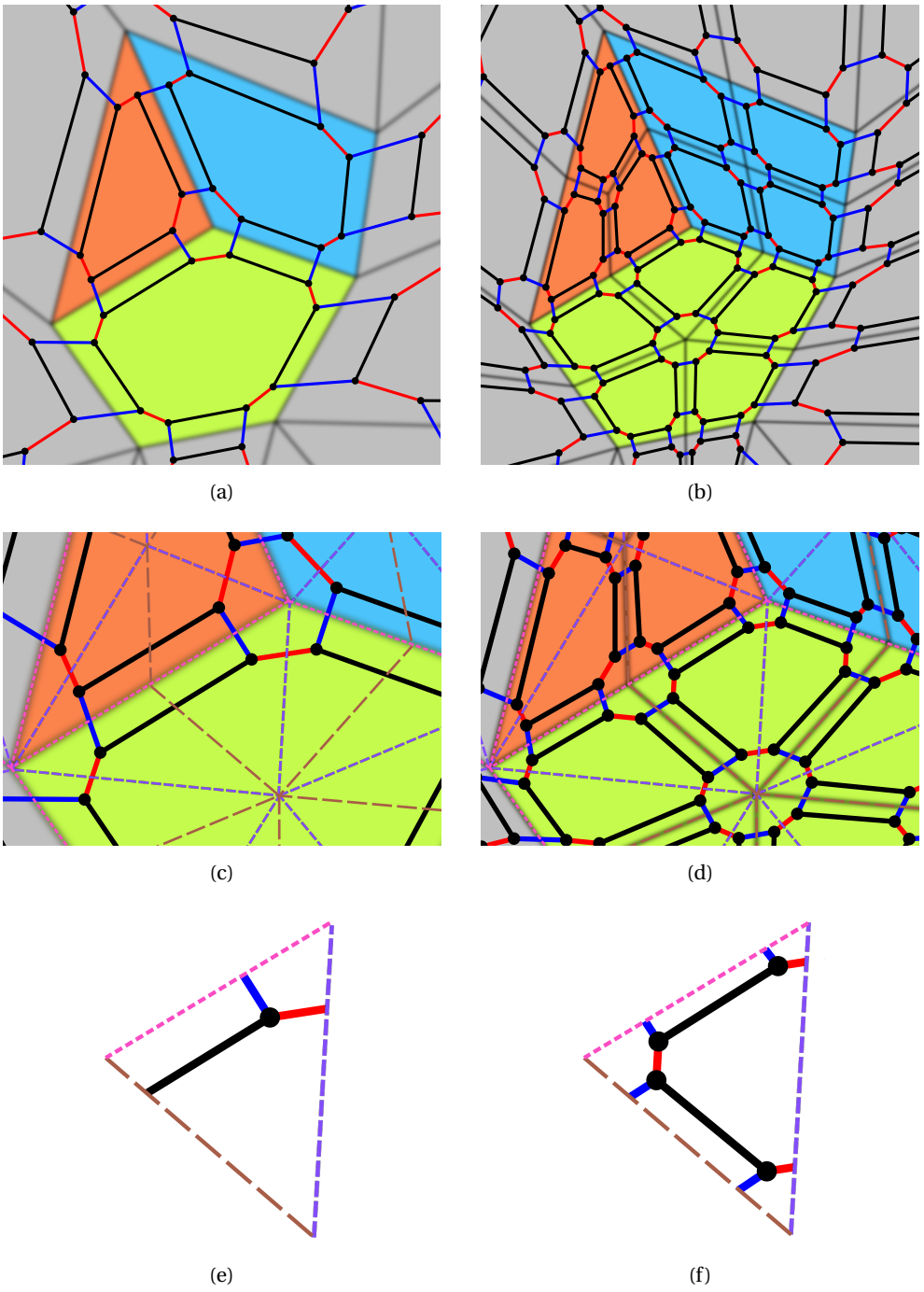


Figure 7.4: Intuition for the inference of an operation: (a) local extraction of a surface, (b) result of the quad subdivision on the extraction, (c) Gmap structure with isolation of the local motifs (brown for 0-links, purple for 1-links, and pink for 2-links), (d) Gmap structure of the result of the operation with isolation of the local motifs, (e) atomic motif before the operation, and (f) atomic motif after the operation.

7.2 Topological folding algorithm

We now present the main contribution of this chapter, which is a mechanism to infer a generic modeling operation from an application example. The objective is to reconstruct the rule scheme that generalizes a specific modeling operation for a given orbit. Rather than directly reconstructing the rule scheme, we first design an algorithm to reconstruct a single graph scheme. Via the joint representation, we can manipulate rules as if they were graphs. Hence, the algorithm extends straightforwardly from graphs to rules, which we will discuss in Section 7.2.4. Recall that a graph

scheme is a graph decorated on each node with an orbit type encoding a relabeling function.

Our algorithm reconstructs a graph scheme $\mathcal{G}$ from a given Gmap G and a given orbit type $\langle o \rangle$ (with $o \subseteq 0..n$). The algorithm works as follows. First, we choose a dart a in the Gmap. We assume that the orbit graph incident to a corresponds to the orbit used for instantiation, i.e., an orbit with the same type as the decoration of the hook h from the graph scheme $\mathcal{G}$. By traversing the Gmap, we then try to reconstruct the graph scheme. Intuitively, the construction of the graph scheme $\mathcal{G}$ consists of locally folding the graph G along its i-links, for all dimensions i of the orbit type $\langle o \rangle$. The folding is then considered up to a relabeling, which must be unraveled while folding the graph. If the construction fails, we start again with another initial dart until a graph scheme is found. The new initial dart is chosen in a pool of unseen darts (see discussion in Section 7.3.2). If all darts have been tried, then no graph scheme exists for the orbit type $\langle o \rangle$.

7.2.1 Notations

In the explanation of the algorithm, we will write:

- h for the hook in the graph scheme $\mathcal{G}$, m for the node of interest, and v for the other nodes.
- Letters from the beginning of the alphabet for the darts in $G\langle o \rangle(a)$, e.g., a, b, c, with a being reserved for the initial dart.
- Letters from the end of the alphabet for generic darts in G, e.g., x.
- d, i and j for the dimensions in $0..n$.
- $\langle o^v \rangle$ for the orbit type used to decorate the node v of $\mathcal{G}$, such that the associated relabeling function is $\langle o \rangle \mapsto \langle o^v \rangle$.
- (b, v) for the dart in G that corresponds to the dart b from the orbit $G\langle o \rangle(a)$ and the node v from the graph scheme $\mathcal{G}$.

7.2.2 Algorithm

We detail the algorithm that constructs a graph scheme for a given connected Gmap G, a given orbit type $\langle o \rangle$, and a chosen dart a of G. The graph scheme $\mathcal{G}$ we seek to construct will have by construction a hook h decorated with $\langle o \rangle$ if it exists. The key parts of the algorithm are the functions `createHook`, `arcExt`, and `buildOrbType` detailed in the following sections. The function `orbType` returns the set of dimensions of the orbit type decorating a node.

As we explain the algorithm (Algorithm 6), we will illustrate it with the folding of a topological cube, i.e., a hexahedron, from one of its vertices, i.e., on the orbit $\langle 1,2 \rangle$. The cube is considered a closed surface and represented with a 2-Gmap. In the figures, nodes are generically named n plus an integer, e.g., $n0$, $n1$, following Jerboa's naming convention.

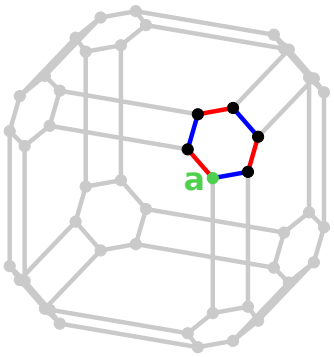
Step 1 (Orbit graph and construction of the hook) - function `createHook`

We build the orbit graph $G\langle o \rangle(a)$ on a in G and initialize the graph scheme $\mathcal{G}$ as a single hook h decorated with the chosen orbit type $\langle o \rangle$. Retrieving the orbit can be achieved via Algorithm 4 (see Chapter 6). $G\langle o \rangle(a)$ is by construction a possible instantiation of $\mathcal{G}$. This step is illustrated in Figure 7.5.

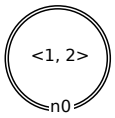
Algorithm 6: Topological folding algorithm

Input: A Gmap G, an orbit type $\langle o \rangle$, and a dart a of G.
Output: The graph scheme $\mathscr{S}$ such that the instantiation $\iota^{\langle o \rangle}(\mathscr{S}, \mathrm{G}\langle o \rangle(a))$ maps (h, a) onto a.

```
1  Q ← empty queue
2  S ← ∅                                                  // Empty graph scheme
3  h ← createHook(S,⟨o⟩)                                  // Construction of the hook (Step 1)
4  enqueue(Q,h)
5  while Q not empty do                                   // Traversal (Step 2)
6      m ← dequeue(Q)
7      foreach d ∈ (0..n)\ orbType(m) do
8          v ← arcExt(G,m,d)      // Extension of the explicit arcs incident to v (Step 2.1)
9          buildOrbType(G,v)      // Construction of the orbit type decorating v (Step 2.2)
10         enqueue(Q,v)
11 return S
```



(a)



(b)

Figure 7.5: Step 1 of the topological folding algorithm: (a) construction of $\mathrm{G}\langle 1,2 \rangle(a)$, and (b) the rule contains a single node marked as a hook and decorated with the orbit $\langle 1,2 \rangle$.

Step 2 (Traversal)

Using a breadth-first traversal of G, we reconstruct a graph scheme $\mathscr{G}$ that yields G by instantiation when (a, h) is mapped onto a. First, the algorithm constructs the explicit arcs incident to the next node (Step 2.1), starting from h. Then, it immediately adds the node decoration (Step 2.2) of each newly created node. Finally, it iterates on a new node. Both steps are realized in a try-and-check approach. Starting from the dart a, or its copy a' associated with the current node, we analyze the status of the links incident to a. For a dimension d, we can find a loop, a link connecting a' to another dart associated with the same node, a link connecting a' to a dart associated with another node, or a link connecting a' to a dart not yet associated with a node. Excluding the loop case, each possibility corresponds to a unique case in the instantiation procedure. Therefore, we assume that the link comes from the corresponding instantiation case. This assumption only constitutes a plausible solution, although necessary to obtain the link incident to a'. We must check that the assumption made also yields the correct links incident to the other darts associated with the same node as a'. If the solution is valid, we resume the traversal of the graph. Otherwise, no solution exists: we found two nodes requiring conflicting treatment in the graph scheme to obtain their incident arcs. In the loop case, it can be created by either the instantiation of an explicit loop or the instantiation of an implicit arc if a has a loop incident for that dimension. Both possibilities are tested, starting from the implicit arcs, as the extension of explicit arcs only occurs on nodes with a

properly defined orbit type. Examples for the case of failures will be presented in Section 7.2.3.

Step 2.1 (Extension of the explicit arcs incident to a node) - function `arcExt`

Given a node m in $\mathcal{G}$, already decorated with an orbit type, we construct its incident arcs. These explicit arcs are labeled by dimensions d not belonging to the orbit type $\langle o^m \rangle$. Thus, their instantiation provides the remaining links incident to the darts of the instance of m.

The arc construction starts with finding an extension of the graph scheme based on the information we can retrieve from (a,m). Then, the extension is verified to be compatible with all the (b,m) for b in $G\langle o \rangle(a)$. The verification is straightforward as the addition of a node m to $\mathcal{G}$ ensures that we can retrieve all the darts (b,m) for b in $G\langle o \rangle(a)$.

Since the algorithm runs on an n-Gmap G, the dart (a,m) has exactly one incident d-link e_d per dimension d in $(0..n) \setminus \langle o^m \rangle$. There are three possibilities for each e_d:

Arc addition e_d is a d-link $(a,m) \overset{d}{\bullet\!\!-\!\!\bullet} (a,m')$ where m' is a node of $\mathcal{G}$. In this case we add a d-arc $m \overset{d}{\bullet\!\!-\!\!\bullet} m'$ in $\mathcal{G}$ (if not already in $\mathcal{G}$ by the extension of m').

Arc failure e_d is a d-link $(a,m) \overset{d}{\bullet\!\!-\!\!\bullet} (b,m')$ where b is a dart of $G\langle o \rangle(a)$ different from a, and m' is a node of $\mathcal{G}$. In this case, the algorithm stops in a failure state because such a link is not constructible using the instantiation mechanism.

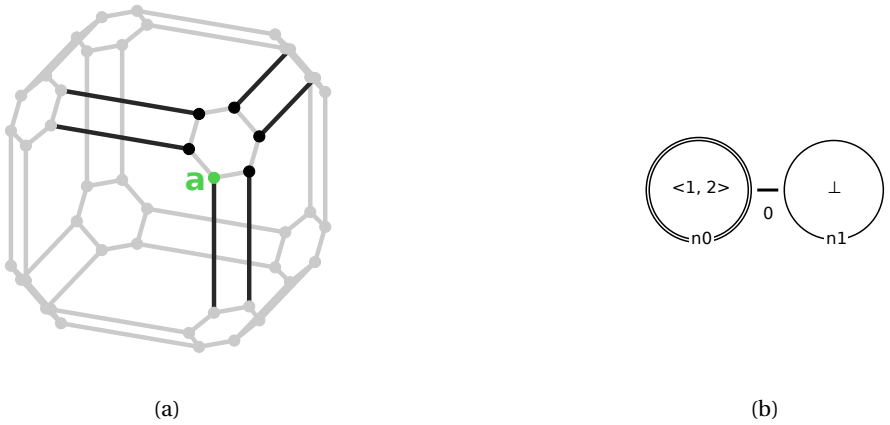
Arc extension e_d is a d-link $(a,m) \overset{d}{\bullet\!\!-\!\!\bullet} x$ where x has not yet been reached by the algorithm. In this case, we add a node v with the decoration $\langle o^v \rangle = \bot$ and a d-arc $m \overset{d}{\bullet\!\!-\!\!\bullet} v$ in $\mathcal{G}$. The symbol $\bot$ serves as a placeholder to signify that the node v has no decoration yet, with the convention that $\bot$ deletes all arcs when instantiated.

Let $\mathcal{G}'$ be the graph scheme obtained after the addition and extension of all the d-arcs for d in $(0..n) \setminus \langle o^m \rangle$ (assuming no failure). Recall that a d-arc in the graph scheme gives rise to one d-link per dart b in $G\langle o \rangle(a)$, each one incident to a dart (b,m). Therefore, we still need to check that the decision made for each dimension is coherent with the links incident to the other darts. In other words, check that each dart (b,m) for b in $G\langle o \rangle(a)$ has an incident d-link with the other extremity corresponding to the same node in $\mathcal{G}'$.

Instantiation failure For each d-arc $m \overset{d}{\bullet\!\!-\!\!\bullet} m'$ in $\mathcal{G}$, if a dart b exists in $G\langle o \rangle(a)$ such that there is no link $(b,m) \overset{d}{\bullet\!\!-\!\!\bullet} (b,m')$ in G, then the algorithm stops in a failure state because such an arc would be constructed using the instantiation mechanism.

Note that the case of 'arc addition' where $m' = m$ covers the addition of loops to the graph scheme $\mathcal{G}$.

Let us resume the illustration of the algorithm started in Figure 7.5. We start on dart a and consider the dimensions not in $\langle 1,2 \rangle$, which leaves the dimension 0. We then check the 0-neighbor of a. Since it corresponds to an unseen dart, we assume the extension of the graph scheme from Figure 7.5b to the graph scheme from Figure 7.6b. The graph scheme is incomplete at this point since node $n1$ has a fake orbit type $\bot$. We then check that all darts in $G\langle 1,2 \rangle(a)$ have an unseen 0-neighbor, which is the case, as illustrated in Figure 7.6a. Thus, the six darts of the vertex $G\langle 1,2 \rangle(a)$ exhibit a 0-link to six distinct darts outside the vertex folded by the hook $n0$. We consider the arc extension valid and proceed with Step 2.2 on $n1$ to add a valid node decoration.



(a) (b)

Figure 7.6: Step 2.1 of the topological folding algorithm: (a) traversal of the link incident to a dart in $G\langle 1,2\rangle(a)$, and (b) extension of the hook's explicit arcs.

Step 2.2 (Construction of the orbit type decorating a node) - function `buildOrbType`

A node m can be added with a fake orbit type in the 'arc extension' case from Step 2.1. The construction of the explicit d-arc assumes that the target darts of the corresponding d-links are the instantiation of the new node m in the graph scheme. The proper orbit type associated with this node needs to be reconstructed. We look for the existence of an orbit type $\langle o^m\rangle$ whose instantiation provides the links that join the darts of the instance of m.

Here again, we first extend the graph scheme $\mathcal{G}$ before checking if the extension is correct. We consider, for each dimension i of $\langle o\rangle$, the i-neighbor b_i of a, i.e., the dart such that $a \overset{i}{\bullet\!\!-\!\!\bullet} b_i$ is in $G\langle o\rangle(a)$. We now distinguish between two possibilities:

Relabeling There is a dimension j in $0..n$ such that a link $(a,m) \overset{j}{\bullet\!\!-\!\!\bullet} (b_i,m)$ exists in G. In this case, we fix the relabeling $i \mapsto j$, i.e., j appears in $\langle o^m\rangle$ at the position of i in $\langle o\rangle$.

Deletion The previous possibility is not fulfilled, i.e., (a,m) and (b_i,m) are not linked. We then fix $i \mapsto \_$, i.e., the symbol "_" appears in $\langle o^m\rangle$ at the position of i in $\langle o\rangle$.

Recall that the orbit type used to decorate m gives rise to arcs based on the relabeling function $(\langle o\rangle \mapsto \langle o^m\rangle)$. Therefore, one still needs to check that the decision made for each implicit arc is coherent with the other darts related to the instantiation of m. In other words, one still needs to check that each i-link between b and c in $G\langle o\rangle(a)$ can be mapped onto a link between (b,m) and (c,m) in G.

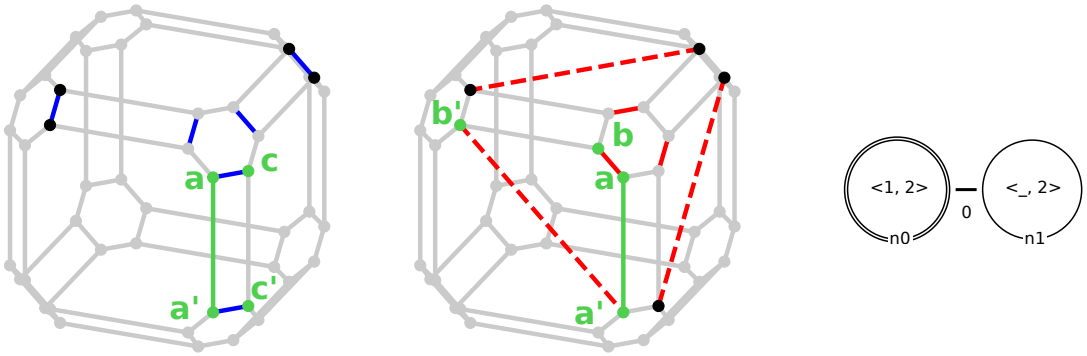
Relabeling failure If the decided relabeling is $i \mapsto j$ and there are two darts b and c in the orbit $G\langle o\rangle(a)$ such that no link $(b,m) \overset{j}{\bullet\!\!-\!\!\bullet} (c,m)$ exists in G, then the algorithm stops in a failure state. Indeed, such a link would be constructed by the instantiation mechanism.

We are identifying the implicit arcs that are preserved, deleted, or relabeled by the new node m. Note that the preservation of an arc is covered by the case 'relabeling,' where $j = i$.

In general, two darts b and c of $G\langle o\rangle(a)$ can be connected by several i-links (i.e., with different i of $\langle o\rangle$). One can then construct several different orbit types. Our algorithm uses a heuristic to construct a plausible relabeling function among all possible ones. To ensure the injectivity of the relabeling function, we ensure that two dimensions i and i' are not mapped onto the same

dimension j. The possible plurality of inferred graph schemes (and therefore rule schemes) is further discussed in Section 7.3.2.

Note that we do not need a failure possibility for the 'deletion' case. Indeed, if the decided relabeling is $i \mapsto \_$, but there are distinct darts b and c of $G\langle o\rangle(a)$ and a dimension j of $0..n$ such that $(b,m) \overset{j}{\bullet\!\!-\!\!\bullet} (c,m)$ is a link of G, then j does not appear in $\langle o^m\rangle$. Since G is a Gmap, the incident arcs constraint ensures the existence of a j-link incident to (a,m). This j-link will bring the addition or creation of a j-arc in the construction of the explicit arcs incident to m (if not directly a failure state). This j-arc will result in an instantiation failure on the darts (b,m) and (c,m).



(a) (b) (c)

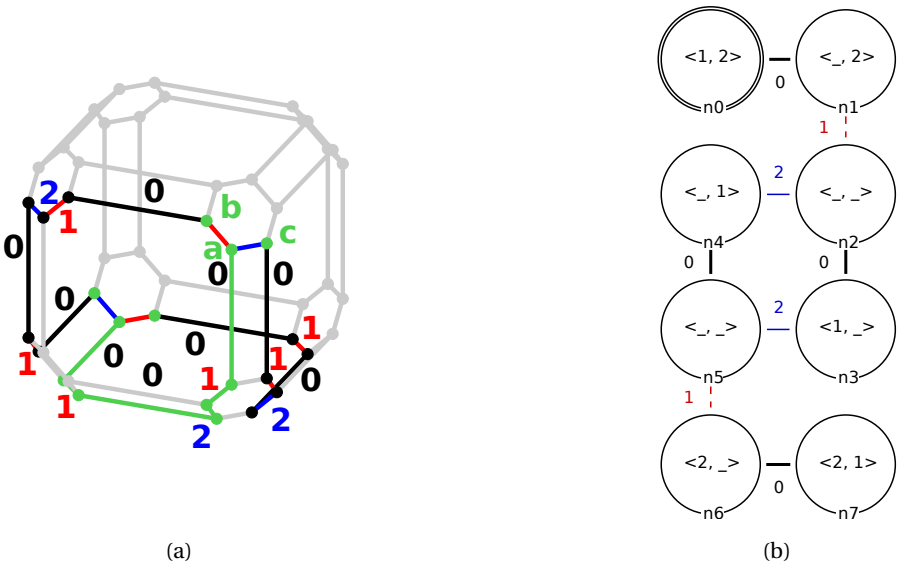
Figure 7.7: Step 2.2 of the topological folding algorithm: (a) analysis of the image of the 2-links from $G\langle 1,2\rangle(a)$ between the darts associated with $n1$, (b) analysis of the image of the 1-links, and (c) construction of the orbit type decorating the node.

In the previous step, illustrated in Figure 7.6, we added a node $n1$ for which we now have to construct an orbit type. We try to construct a relabeling of the 1 and 2 links of the initial orbit graph on the set of darts associated with $n1$. We start on dart a', i.e., the copy of dart a in the set of darts associated with node $n1$ by the arc extension (Step 2.1). We check, for each dimension $i \in \langle 1,2\rangle$, if a link exists between a' and the copy of the i-neighbor of a in the set of darts associated with node $n1$. For the dimension 2, we consider the Figure 7.7a. Dart c is the 2-neighbor of a. The copy of c in the set of darts associated with node $n1$ is c'. Following the notation in the explanation, c' correspond to $(c,n1)$. As illustrated in Figure 7.7a, G has a link $a' \bullet\!\!-\!\!\bullet c'$. Therefore, we assume the relabeling $2 \mapsto 2$. We then check if the relabeling is coherent with the other darts associated with $n1$. The relabeling is indeed coherent, as shown in Figure 7.7a. For the dimension 1, we consider the Figure 7.7b. Dart b is the 1-neighbor of a and b' is its copy in the set of darts associated with $n1$. No link exists between darts a' and b', thus the relabeling $1 \mapsto \_$ is assumed. This deletion is coherent with the other darts associated with $n1$, as shown in Figure 7.7a. Therefore, we decorate node $n1$ with the orbit type $\langle \_,2\rangle$ and obtain the partial graph scheme given in Figure 7.7c.

Termination

If the traversal terminates without failure, we obtain a topologically correct graph scheme $\mathscr{G}$ for the chosen dart. Note that the traversal necessarily stops when all darts have been folded.

On the cube, the algorithm terminates with the construction of the orbit of the last node, node $n7$. This node corresponds to the vertex opposite to the initial vertex, as shown in Figure 7.8a. Following the exploration path 012010, we reach the darts of the opposite vertex, relabeling the or-



(a) (b)

Figure 7.8: Termination of the topological folding algorithm: (a) the algorithm terminates with the vertex opposite to $G\langle 1,2\rangle(a)$, and (b) result of the algorithm.

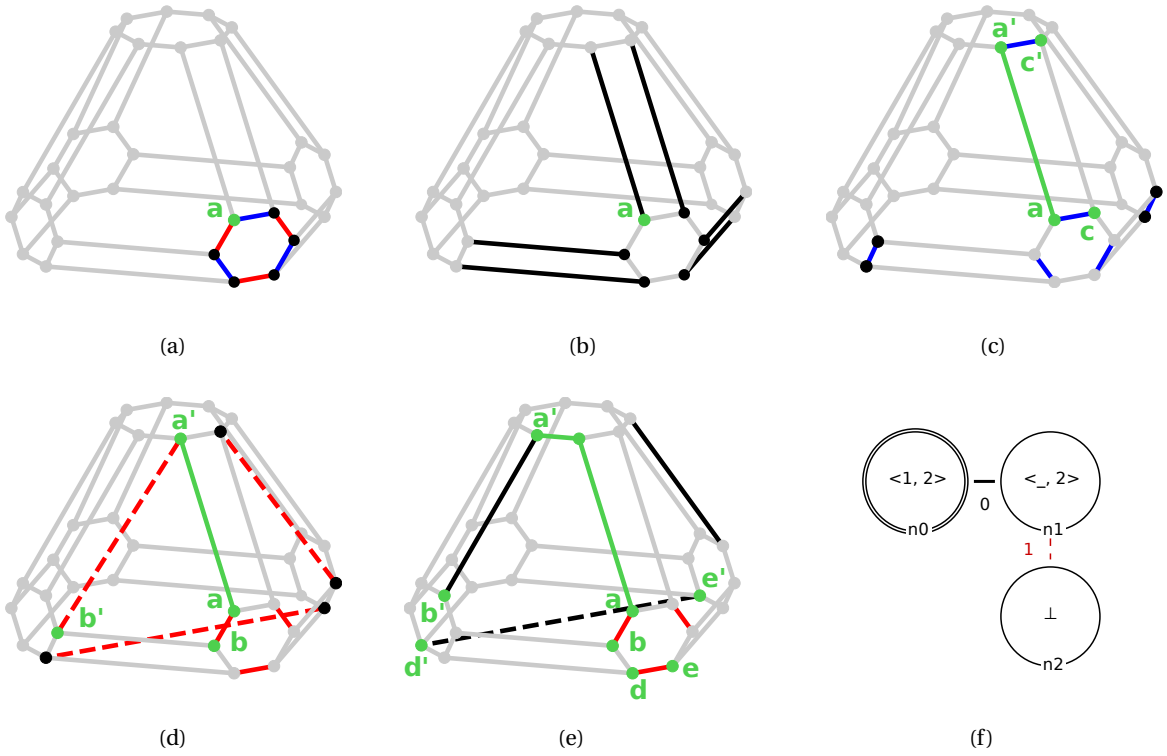
bit $G\langle 1,2\rangle(a)$ by the function $\langle 1,2\rangle \mapsto \langle 2,1\rangle$. The resulting graph scheme is given in Figure 7.8b.

7.2.3 Counterexamples

Before dealing with the analysis of the algorithm, we present the two prominent cases where the algorithm halts in a failure state. Step 1 corresponds to an unrestricted graph traversal and will always succeed. Step 2.1 concerns the extension of arcs incident to a node, while Step 2.2 deals with the orbit type decorating a node. Recall from Chapter 6 that the instantiation of a node with an orbit graph is the application of a relabeling function while the instantiation of an arc corresponds to its transformation in as many links as darts in the orbit. Both kinds of instantiation can be compromised if an incoherent assumption is derived from the analysis of the copy of a for a given node.

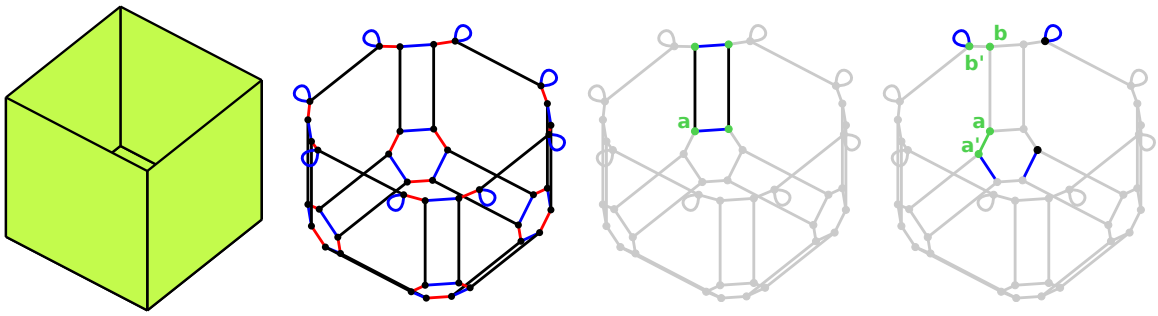
For the construction of the relabeling function associated with a node, the conditions ensure that the relabeling found is coherent for all darts associated with the node. The algorithm will fail to fold a pyramid from a corner of the base, i.e., not the apex. Figures 7.9a, 7.9b, 7.9c, and 7.9d display the same steps as in the folding of the cube, meaning that the partial graph schemes of Figures 7.5b, 7.6b and 7.7c will correctly be built by the algorithm. However, as illustrated in Figure 7.9e, the algorithm will fail. The partial graph scheme at this moment is provided in Figure 7.9f. When building the relabeling, we identify a 0-link between darts a' and b'. We can thus check if the relabeling $1 \mapsto 0$ is coherent with the other darts. The 1-arc between d and e is mapped onto a non-arc between d' and e'. Therefore, the algorithm fails, meaning that no folding can be obtained from a as the initial dart and $\langle 0,1\rangle$ as the orbit type.

For the extension of an arc, the condition ensures that all links folded into the arc have the same status. These links can either be loops or links to darts that all correspond to the same node (coherently with the image from the initial orbit graph) or links to darts not yet seen (checked for relabeling afterward). Whenever the incident d-links for a given dimension d outside the orbit type fall into different categories, an explicit arc cannot be added. In this case, the algorithm stops. For instance, let us consider the object of Figure 7.10a. The object consists of a cube where the top



(a) (b) (c) (d) (e) (f)

Figure 7.9: Steps of the folding algorithm from a non-apex vertex of the pyramid. (a) Step 1 - construction of $G\langle 1,2\rangle(a)$, (a) Step 2.1 - traversal of the link incident to a dart in $G\langle 1,2\rangle(a)$, (a) and (b) Step 2.2 - analysis of the image of the 2- and 1-links from $G\langle 1,2\rangle(a)$ between the darts associated with $n1$, (e) failure in Step 2.2, inconclusive analysis of the image of the 1-links between the darts associated with $n2$, and, (f) partial scheme before failure of the algorithm.



(a) (b) (c) (d)

Figure 7.10: Folding algorithm on a vertical edge of the box: (a) the object, (b) its Gmap representation, (c) darts and links associated to the hook, (d) the algorithm stops on an inconclusive output.

face has been removed. Its Gmap representation is depicted in Figure 7.10b. If we fold it along one of the vertical edges, the algorithm fails. Indeed, with the orbit type $\langle 0,2\rangle$ and the dart a in Figure 7.10c, we reach the state of Figure 7.10d after traversing and folding along the 1-links. Following the construction of Step 2.1, the algorithm considers the neighborhood of a' and finds a non-loop 2-link. Therefore, a non-loop explicit 2-arc should be added to the graph scheme. However, this extension is incompatible with dart b', which has a 2-loop. The algorithm cannot proceed. Thus, no folding is obtainable with a as the initial dart and $\langle 0,2\rangle$ as the orbit type.

7.2.4 Generalization to a rule scheme

The algorithm presented in the previous section allows deducing a folded representation of a Gmap from an orbit type and a dart. From Chapter 3 and the discussion about rule isomorphisms in Chapter 6, we know that a rule can be represented as a graph via the joint representation of rules. This representation builds a graph $\kappa(L,R)$ from two a rule $L \rightharpoonup R$ by considering the disjoint union of the graphs L and R and adding a κ-arc between preserved nodes. For instance, the graph $\kappa(L,R)$ constructed from the rule of Figure 7.2a is given in Figure 7.11a. The κ-arcs are drawn in pink; the original cube is enlarged while the modified cube is shrunk. This representation is valid for both instantiated rules and rule schemes. Therefore, we propose to construct this joint representation from the before and the after instances of the representative example describing the operation. We then use the topological folding algorithm on the joint representation to obtain a joint representation of a rule scheme. If we carry on the folding of the cube from a vertex but on the joint representation with the subdivided cube, i.e., on the graph of Figure 7.11a, we obtain the graph of Figure 7.11b. The green nodes are associated with darts from the before instance, while the blue nodes are associated with darts from the after instance.

When folding a joint representation, the algorithm needs to be slightly altered, and additional assumptions should be made:

- We memorize whether a dart belongs to L or R before running the algorithm.
- We assume the graph $\kappa(L,R)$ to be connected.
- We choose a dart from L as input for the algorithm.
- We never fold a κ-link. Therefore, κ is not considered to construct a relabeling function $[\langle o \rangle \mapsto \langle o^m \rangle]$ and κ-arcs can only occur as explicit arcs in the output of the algorithm.

The desired rule scheme is obtained by removing the κ-arcs from the output graph. The graph is split into two graphs based on the status of its associated darts. Note that a node necessarily has all its associated darts either in the before or the after instance since κ-links are not folded. This removal yields two graph schemes corresponding respectively to the rule scheme's left and right-hand sides. Finally, the inferred rule scheme is checked against the conditions for the preservation of the topological constraints (conditions from Chapter 4 adapted to orbit-decorated rule schemes). This verification is necessary as invalid rule schemes might be reconstructed, which we will discuss more thoroughly when analyzing the algorithm in Section 7.5.

When considering the folding of rules rather than Gmaps, we need to consider two edge cases: creation and suppression operations. In the former case, the rule's before instance is empty, meaning that the rule cannot be folded. Thus, creation rules only have a right-hand side corresponding to the after instance. In the latter case, the topological folding algorithm can still be realized. We obtain deletion operations parameterized by orbit types, e.g., the deletion of a surface or a prism.

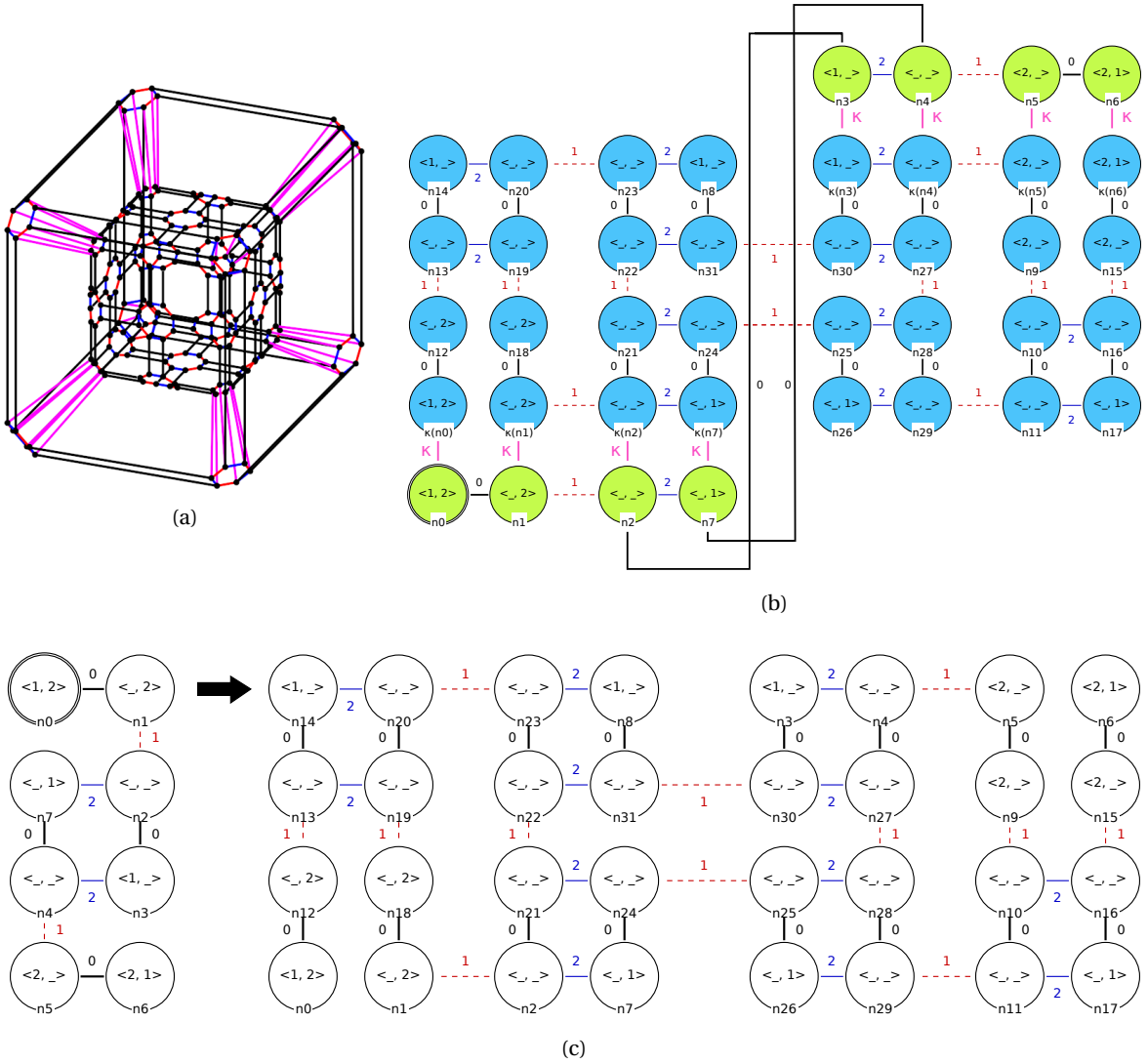


Figure 7.11: Folding the quad subdivision on the joint representation of the cube: (a) the joint representation $\kappa(L,R)$ for quad subdivision of the cube, green nodes belong to the rule's left-hand side and blue nodes to the right-hand side. (b) the joint representation of the rule scheme retrieved by the topological folding algorithm, and (c) the final rule scheme.

7.3 Jerboa Studio

Historically, the viewer and the rule editor are two separate tools in Jerboa. The editor is used to specify the operations and embeddings that constitute a geometric modeler, while the viewer allows for the manipulation of geometric objects. This separation was motivated by the distinction between the users of each tool. The editor was meant to be used by a specialist in geometric modeling as a solution to build software for a domain expert who would use the viewer. The inference of geometric modeling operations blurs this separation. Indeed, we may need to manipulate the inferred operations, e.g., for edition or comparison. Besides, we may also need to apply the inferred operations without regenerating the whole modeler. To this end, we have incorporated both the previously presented viewer and the rule editor in a fully integrated tool called *Jerboa Studio*. Therefore, all features of the rule editor are freely available. For instance, we can reuse the syntactic checker available in the rule editor to check for the consistency of the inferred operations. Note that modeling operations have a distinct internal representation in the viewer and the editor. The

operations are compiled Java code in the viewer, while they are essentially stored as graphs in the rule editor. We added a third representation of rules for the inferred operations, which stores the necessary information to be used in both parts of the tool.

7.3.1 A framework to infer, edit and apply geometric modeling operations

For the inference of geometric modeling operations, we need to provide two instances of a representative example of an operation. Therefore, we adapted Jerboa's default viewer to obtain the viewer depicted in Figure 7.12. In our dedicated viewer, we display two objects. The left one represents the before instance, and the right one represents the after instance. We also have to offer a solution to provide the partial morphisms describing the instantiated operation. At the bottom of the interface, the user can select the darts that should be taken into account to build the operation. The selection delimits the scope of the operation and has to be realized in both objects. We offer the possibility to add all currently selected darts to the set of considered darts for the operation. We can therefore exploit the various selection mechanisms to ease the addition of darts, e.g., manual selection, orbit selection, and dimension traversals. We consider any link between two selected darts as part of the transformation. In practice, this selection mechanism also simplifies the inference mechanism. Indeed, we can design operations by changing the set of considered darts between the before and after instances without modifying the object.

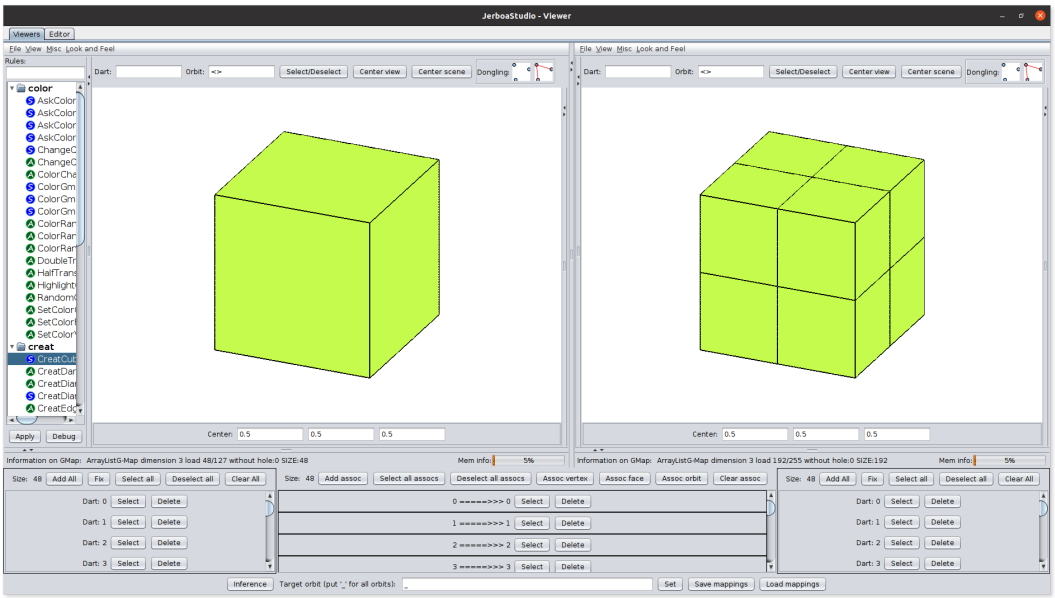


Figure 7.12: Viewer for the inference of operations.

The user should also provide the mapping of the preserved darts from the left pattern to the right pattern. In our first iteration of the tool, the mapping was provided implicitly. The user would modify the object directly into a single viewer and take two snapshots before and after the modifications. We would then build the mapping from identifiers of the darts present in both snapshots. Improving on this initial approach, the user may now provide an empty mapping and let Jerboa retrieve it. However, this solution had several drawbacks. The initial instance would no longer be accessible after being modified to obtain the target instance. Besides, some operations may require a non-identify mapping, especially if the objects were obtained by other means and imported. Therefore, we also allow for explicit mapping, manually provided by the user. In this

case, the user builds left-to-right associations on darts, but we also allow the mapping of orbits to simplify the process. Before running the algorithm, the user can specify the desired orbit type or leave the parameter empty to try all possible orbits. They can also specify additional parameters to twist the algorithm's behavior, as shown in Figure 7.13.

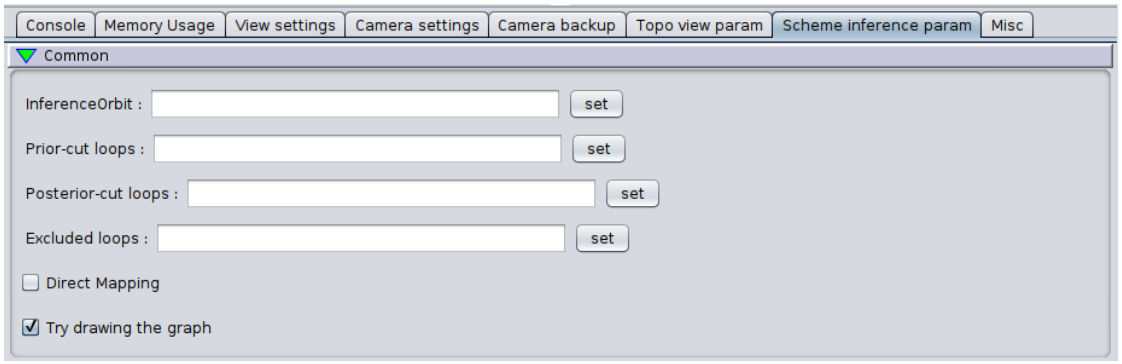


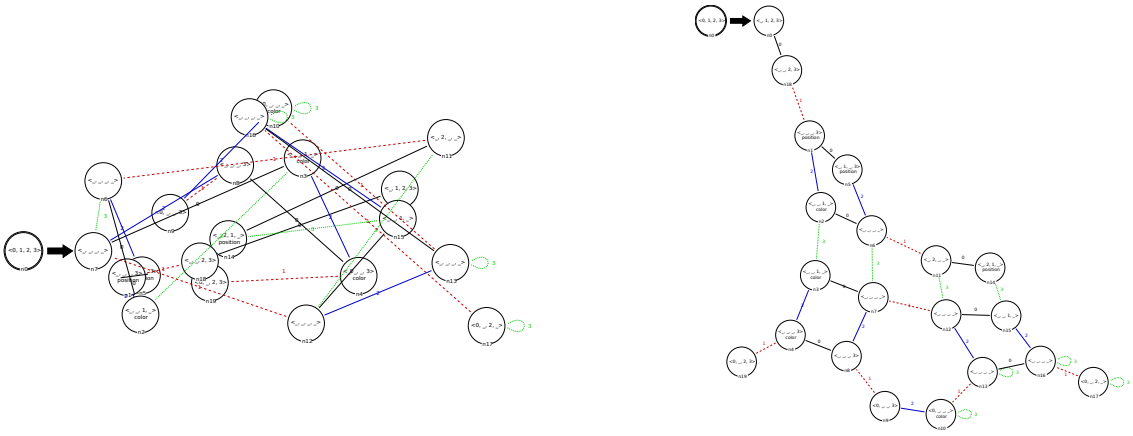
Figure 7.13: Parameters for the topological folding algorithm.

- `InferenceOrbit` is the orbit type used as input of the algorithm.
- `Prior-cut loops` allows removing loops incident to preserved darts in the joint representation (per dimension).
- `Posterior-cut loops` allows removing loops on inferred rule scheme (per dimension).
- `Excluded loops` states which dimensions should be considered explicit loops when both the explicit loop and the implicit arc as relabeling of loops are correct.
- `Direct Mapping` states whether the construction of the orbit types decorating the nodes should be realized from the hook to the node or reversely. We primarily use it for debugging purposes.
- `Try drawing the graph` states whether a graph display algorithm should be used on the inferred rule schemes. The drawing of graphs is a research field that we will not detail. We refer the curious reader to the survey done in [77] or [32, Chap. 3]. We implemented the two most commonly used algorithms, i.e., Kamada and Kawai [108] and Fruchterman and Reingold [72] with some improvements from [107] used in the tool Gephi. Illustrations with and without the automatic layout are given in Figure 7.14. We offer the possibility to disable it, as drawing large graphs can be computationally expensive.

Jerboa Studio provides a framework to infer, edit and apply geometric modeling operations, thanks to the topological folding algorithm. Jerboa Studio is available on Jerboa's webpage, accompanied by an application for geology purposes[2] and an application for subdivision schemes.[3]

[2] Link to website (last consulted on September 24th, 2022): `http://xlim-sic.labo.univ-poitiers.fr/jerboa/doc/inference-of-topological-operations-illustration-in-geology/`.

[3] Link to website (last consulted on September 24th, 2022): `http://xlim-sic.labo.univ-poitiers.fr/jerboa/doc/topological-inference-for-subdivision-schemes/`.



(a) (b)

Figure 7.14: A graph layout algorithm helps visualize the inferred rule: (a) without, and (b) with a layout computation.

7.3.2 Practical discussion about the implementation of the algorithm

In Section 7.2, we explained that the topological folding algorithm takes as input two Gmaps, a dart mapping between these Gmaps, a dart from the initial Gmap, and an orbit type while producing a rule scheme as output. Intuitively, the algorithm acts on the joint representation of the Gmaps induced by the mapping. However, the practical reality is more complex, and we now discuss some distinctions between theory and practice.

Local rules and internal representation

In Section 7.2, we illustrated the topological folding algorithm with the quad subdivision. The quad subdivision is a global operation applied on a mesh to refine it, meaning the whole Gmaps were considered part of the operation. However, operations may only modify the object locally. For local operations, only parts of the Gmaps should be considered, i.e., the scope of the operation needs to be delimited. Handling local operations is the motivation of the selection step described in Section 7.1.2. When considering modeling operations that only locally modify an object, we lose the incident arcs property on the graph, meaning that a dart may not have a link for a given dimension. This property does not apply to the κ-links encoding the mapping between the two instances. Therefore, the data structure manipulated by the algorithm is neither a Gmap nor two Gmaps linked by κ-arcs, but the joint representation, allowing for undefined neighbors (see Definition 36 in Chapter 3 and the discussion of Section 7.3.3). We obtain an efficient data structure for graph traversals: for orbit retrievals via Algorithm 4 from Chapter 6, and for the present topological folding algorithm. This representation is also used for the algorithm's output, where each node stores an orbit type as decoration and a reference to its associated darts in the initial graph. We chose this representation rather than the one used in the editor to simplify pre-treatments and post-treatments on the algorithm. For instance, we may want to remove loops on preserved elements to avoid restricting the operations to free cells (for a dimension). The removal of loops can be realized either as a treatment on the Gmaps or as a post-treatment on the rule scheme. Another post-treatment comes from the possible plurality of inferred operations, which we discuss next.

Topological consistency

The folding algorithm provides a solution to generalize an operation from a representative example. This generalization is obtained by finding symmetries in the operation (up to relabeling). Such a process tries to reverse the instantiation of a graph scheme. However, the generalization of an operation might be topologically incorrect, i.e., not fulfilling the topological conditions from Chapter 4, which ensure that applying a rule scheme to a Gmap always provides a Gmap. Intuitively, the inconsistent rules come from incoherent generalizations, i.e., from the orbit used for the inference, the section, or the mapping. However, the topological folding algorithm might still produce an output. We illustrate such a possibility in Figure 7.15. The before and after instances respectively consist of triangular and hexagonal faces. The complete topological structures are given. These are 2D objects, and we can try to infer the associated operation with the most general orbit, i.e., the orbit $\langle 0,1,2\rangle$. We now consider two possible mappings indicated with κ-links.

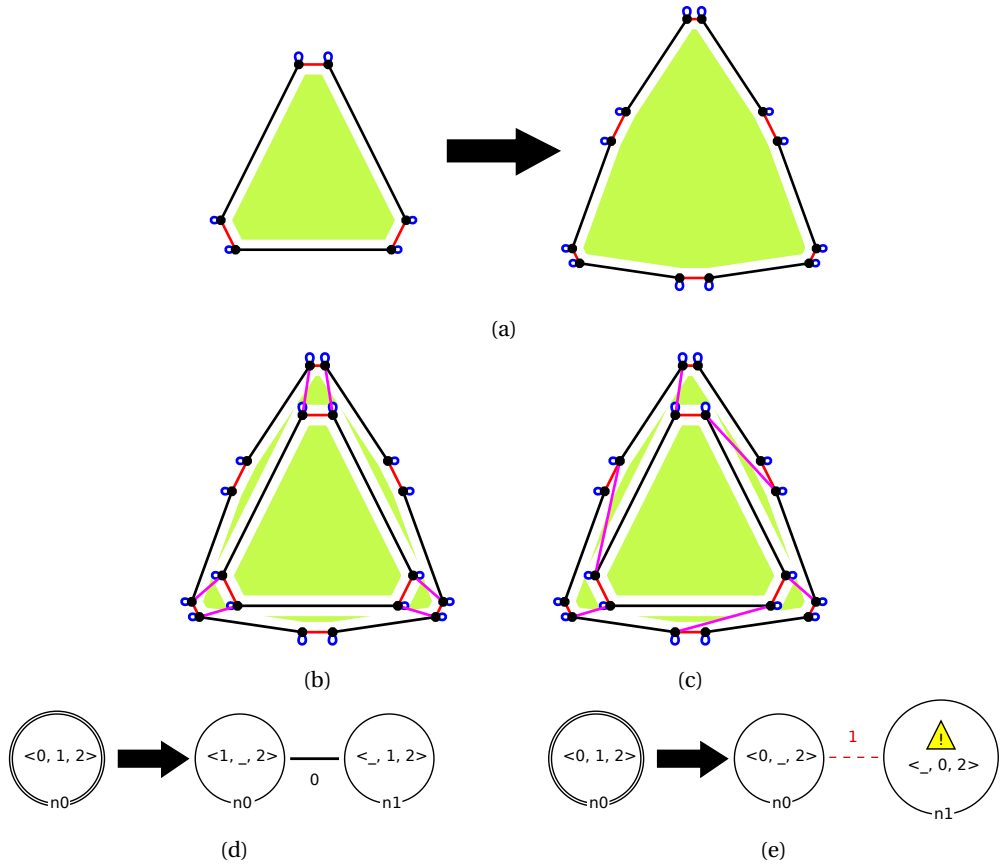


Figure 7.15: Topological consistency: (a) the before instance is a triangle, and the after instance is a hexagon, (b) the mapping induces an edge split, (c) the mapping induces a vertex split, (d) folded rule for the mapping of Figure (b), and (e) incorrect rule obtained with the mapping of Figure (c).

In Figure 7.15b, the mapping induces an edge split. The operation preserves each vertex, and the new darts split the edges by inserting new vertices. This mapping yields the rule of Figure 7.15d, detected as well-formed by the syntactic analyzer. When applied on a surface, this operation inserts a vertex to split each edge.

In Figure 7.15c, the mapping induces a vertex split by preserving the edges. Although valid on a curve, this operation becomes topologically incorrect when applied on a surface where 2-links are not loops. For instance, inserting an edge in each vertex of a cube is impossible. However,

the topological folding algorithm can produce a folded representation of this operation from the two faces. The rule is given in Figure 7.15e. Via the syntax checker, we find that this operation would break the topological consistency, as illustrated by the flag on node $n1$. Therefore, this rule is discarded, and the algorithm outputs that no operation can be inferred.

Note that the syntactic analyzer works on the representation of rule schemes used in the rule editor. Therefore, the joint representation of the inferred rule, i.e., the algorithm's output, is first translated into the representation of the rule editor.

Plurality of inferred operations

Intuitively, the topological folding algorithm deterministically folds a graph with a given orbit type and an initial dart. Although the algorithm is deterministic, several rule schemes might be valid for a given input. This plurality comes from the dichotomy in the treatments of loops, either as explicit loops or as implicit arcs relabeling loops, already mentioned in Sections 7.1 and 7.2. We choose from the two possibilities using the parameters presented in Figure 7.13, practically solving the non-determinism of the inference.

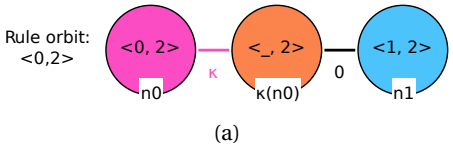
Besides, we might obtain a different rule scheme when starting from a different dart or with another orbit type. We will obtain equivalent and non-equivalent rules when trying all orbit types and initial darts. For instance, all rules on the empty orbit $\langle\rangle$ are isomorphic up to the choice of the hook(s). Rather than running all computations and pruning the set of solutions afterward, we implemented a mechanism marking the darts that would yield a rule already obtained or fail to result in a rule. Each dart stores one mark per orbit type, and we try the algorithm again until all darts are marked with all orbit types. We might obtain the same rule several times based on the symmetry of the initial and target objects. To discard duplicated rules, we use a rule isomorphism algorithm (Algorithm 7) which we present next.

7.3.3 Isomorphism of rule schemes

When inferring operations starting from all possible darts with all possible orbit types, the topological folding algorithm generates a set of rules compatible with the provided example. However, based on the topological symmetries of the object, isomorphic rules may be generated. Therefore, comparing rules provides a solution to discard the duplicated ones. Since inferred rules will mostly be non-isomorphic (with a few exceptions), it is desirable to quickly state whether two rules are non-isomorphic, even if the complete resolution for isomorphic rules may be slightly longer. It should also be considered that comparing rules does not mean comparing their left-hand and right-hand sides separately. The node mapping from the left-hand to the right-hand side matters.

Nonetheless, we propose to solve rule isomorphism as a graph isomorphism problem. The general graph isomorphism problem and its specialization in orbit isomorphism have already been discussed in Chapter 6. To represent a rule as a graph, we use the joint representation of rules, introduced in Chapter 3 (see Section 3.3.2). Intuitively, from a rule $r = \mathrm{L} \hookrightarrow \mathrm{R}$, we build a graph $\kappa(\mathrm{L},\mathrm{R})$ as the disjoint union of L and R, plus arcs labeled by κ (a fresh symbol) connecting both copies of the preserved nodes.

Figure 7.16a illustrates the joint representation of the rule schemes from Figure 6.11b. This is the joint representation for the operation of vertex insertion on both free and sewn non-degenerated edges, which was already discussed in Chapter 3. The right-hand side node $n0$ has been renamed



(a)

Node	Node signature
$n0$	$\langle 0,2\rangle 0001$
$\kappa(n0)$	$\langle \_,2\rangle 1001$
$n1$	$\langle 1,2\rangle 1000$

(b)

Figure 7.16: Representation of a rule scheme for isomorphism computation: (a) joint representation of the rule schemes from Figure 6.11b, and (b) associated node signatures.

$\kappa(n0)$ to distinguish it from its left-hand side counterpart. A κ-arc between nodes $n0$ and $\kappa(n0)$ has been added to signify that both nodes correspond to the same preserved node.

In practice, the κ-arcs are also non-oriented to ensure that a graph traversal of $\kappa(L,R)$ can move back and forth between its L part and its R part. This non-orientation means that a connected component of $\kappa(L,R)$ will be fully traversed, even if its L part or R part are not connected. Therefore, we add a mark (typically a boolean value) on the nodes to clarify whether they come from the rule's left- or right-hand side. In the graph $\kappa(L,R)$, every node admits at most one i-neighbor per dimension, including κ. Some nodes may lack an i-neighbor because i appears as an implicit arc in the orbit type decoration of the node or because the operation does not modify the i-links in the Gmap. Nevertheless, the property of 'at most one arc' means we can still realize an isomorphism computation from a joint traversal.

Compared to the computation of orbit isomorphism (Section 6.2.4), we need to assume two changes. First, the nodes are decorated with orbit types, and the isomorphism can only map nodes with the same decoration. Secondly, we no longer have a starting point in the graph to compute the traversal. Because of these two changes, we perform a precomputation on the graph based on node invariants [128]. Node invariants enable the partial comparison of graphs without building their complete canonical label. A node invariant is a function ϕ such that if $i: G \to H$ is a graph isomorphism, then for all nodes v of G, $\phi(v) = \phi(i(v))$. The converse does not hold. A graph morphism $m: G \to H$ such that for all nodes v of G $\phi(v) = \phi(i(v))$ is not necessarily an isomorphism. An example of such a node invariant might be the node degree, i.e., its number of neighbors. If two graphs are isomorphic, the bijection necessarily maps a node to a node of the same degree. However, a node mapping that preserves the degree is not necessarily an isomorphism. Node invariants can be used in two ways. At a coarse level, if the two graphs do not have the same distribution of values for a node invariant, they are not isomorphic. At a fine level, when looking for a mapping candidate for a given node, we can restrict the search to the nodes with the same value for a node invariant. For the rule isomorphism problem, rather than picking a random node and trying every possible node in the joint representation of the other rule or building a full signature as proposed in [79], we provide a node signature used as node invariant.

Definition 90 (Node signature)**.** *Let $\kappa(\mathscr{L},\mathscr{R})$ be the joint representation of an n-rule scheme $\mathscr{L} \hookrightarrow \mathscr{R}$. The* node signature *of a node v from $\kappa(\mathscr{L},\mathscr{R})$ is the word $\langle o^v\rangle\delta_0\ldots\delta_n\delta_\kappa$ where $\langle o^v\rangle$ is the orbit type decorating v and for $i \in 0..n \cup \{\kappa\}$, $\delta_i = 0$ if v has no i-neighbor, 1 if v as a i-neighbor different from itself and 2 if v is its own i-neighbor.*

The node signatures of the rule scheme from Figure 7.16a are given in Table 7.16b. Node $n0$ has the node signature $\langle 0,2\rangle 0001$: it has $\langle 0,2\rangle$ as the orbit type decoration, no 0-neighbor, no 1-neighbor, no 2-neighbor, and $\kappa(n0)$ as κ-neighbor.

Computing a node signature only requires accessing information directly available from the node and can therefore be computed by iterating over the collection of nodes. After computing the node signatures of a graph, we store them in two (hash) maps, one that associates each signature with all the nodes that share this signature and another one that associates each node with its signature. We can use these maps to perform a series of computations.

- The function `compareSig(G,H)` compares the distribution of signatures for two graphs G and H.
- The function `minSig(G)` returns the signature with the fewest nodes. If several signatures have the same number of nodes, one is returned at random.
- The function `getSig(G,sig)` returns the collection of nodes for a signature `sig`.
- The function `nodeSig(G,node)` returns the signature of a node `node`.
- Given a signature `sig`, `sig.orb` is the orbit part of the signature and $\texttt{sig.}\delta_i$ is the value of δ_i in the signature (for $i \in 0..n \cup \{\kappa\}$).

From these functions, we can derive a rule isomorphism algorithm (Algorithm 7) for connected joint representations. This algorithm starts with the comparison of the distribution of node signatures. We can stop if the distributions differ because the graphs are not isomorphic. Otherwise, we try to build a node mapping. Such a mapping must associate nodes with the same signature. Therefore, we choose the signature with the fewest nodes and then try isomorphisms by associating pairs from this signature. We choose one node with this signature in the first graph. Then, we try every node with the same signature in the other graph as a candidate for an initial mapping. For each candidate, we run a joint graph traversal similar to the orbit isomorphism algorithm, which also checks that associated nodes have the same signature. From the node invariant property and the existence of at most one neighbor per dimension, we know that if no isomorphism has been found with starting pairs from the chosen node signature, no isomorphism exists. Exploiting the node signatures means that the algorithm will perform at most s joint traversals, where s is the size of the collection associated with the chosen signature. Thus, the algorithm runs in time $O(s|\mathrm{V}|)$ where |V| is the number of nodes in the joint representation. Indeed, we have the following time complexities.

- The computation of the node signature is an iteration on the collection of nodes and performs in time $O(|\mathrm{V}|)$.
- The distribution comparison runs in time $O(|\mathrm{S}|) = O(|\mathrm{V}|)$ where |S| is the number of signatures.
- The computation of the minimal signature in time $O(|\mathrm{S}|) = O(|\mathrm{V}|)$ (or $O(1)$ if the minimal signature is stored while computing the signatures).
- Accessing the collection of nodes for a signature and the signature of a node is constant time $O(1)$.

Algorithm 7: Rule isomorphism.

Input: Two graphs G and H, joint representation of rule schemes, enhanced with node signature.
Output: A node mapping from G to H is G, and H are isomorphic, null otherwise.

```
1  Function ruleIsomorphic(G, H) :
2      if not compareSig (G,H) then                          // Node signature distribution
3          return null
4      sig ← minSig (G)                          // Signature with the smallest number of nodes
5      nodesG ← getSig (G,sig)                          // Nodes with the signature 'sig' in G
6      v ← random (nodesG)                                 // Any node from the collection
7      nodesH ← getSig (H,sig)                          // Nodes with the signature 'sig' in H
8      foreach v' ∈ nodesH do                                                  // Try each pair
           // Init data for building the isomorphism
9          mapd ← empty map
10         mapr ← empty map
11         Q ← empty queue
12         enqueue (Q,v)
13         put (mapd,v,v')
14         put (mapr,v',v)
15         while Q not empty do
16             u ← dequeue (Q)
17             u' ← get (mapd,u)                  // Node associated with u in the isomorphism
18             sigu ← nodeSig (G,u)                 // Signature of u (and by construction u')
19             foreach i ∈ (0..n ∪ κ) \ sigu.orb do                    // Potential explicit arcs
20                 if Sigu.δ_i = 1 then                              // Neighbor from explicit arc
21                     x ← u@i
22                     x' ← u'@i
23                     if x ∈ mapd and get (mapd,x) ≠ x' then      // Non-injectivity from H to G
24                         breakwhile                                  // Start again with a new pair
25                     if x' ∈ mapr and get (mapr,x') ≠ x then     // Non-injectivity from G to H
26                         breakwhile
27                     if nodeSig (G,x) ≠ nodeSig (H,x') then          // Different signatures
28                         breakwhile
                       // Keep building the mapping
29                     put (mapd,x,x')
30                     put (mapr,x',x)
31                     enqueue (Q,x)
32         return mapd
33     return null
```

We can run the algorithm on pairs of connected components for disconnected joint representation of rule schemes. The time complexity increases exponentially with the number of connected components, but comparing the distribution of node signatures should discard many such pairs in practice. In practice, since the topological folding algorithm works with a connected joint representation of the before and after instances, it only produces connected joint representation of rule schemes. Therefore, the joint representation is always connected for the cases where we use this algorithm.

7.4 Results

We retrieved several operations as a validation of our topological folding algorithm. We tried operations with high regularities, mostly subdivision schemes, and operations that had been conceived previously in other projects done with Jerboa, although some operations might fall in both cases. Both cases benefit that a ground truth existed before the inference, meaning we could compare the solution found to the ground truth. We successfully inferred the topological part of the already existing operation in all cases. Relevant geometrical information was manually added to the inferred rule to display the objects and illustrate our results. These two case studies each support another motivation. For subdivision schemes, we can change the orbit type used as input to infer more or less generic operations. For instance, we can trivially generalize all surface refinement procedures to 3D objects by inferring with the orbit type $\langle 0,1,2,3\rangle$. We obtain operations that refine all surfaces of a connected set of volumes. For already conceived operations, we tried operations that took hours to properly design in Jerboa as a baseline for 'hard-to-conceive' operations[4]. We also inferred operations not already available in Jerboa, such as the Powell-Sabin 6-split operation.

7.4.1 Applications for geology and terrain modeling

The content of this section has been presented in [137]. Geologists study the subsoil and its geometry for various purposes, e.g., seismic risk prevention or oil reservoir detection. The geometrical construction of the subsoil is a two-step process. First, measurements are taken in several spots. Secondly, geometry is deduced from these measurements and the knowledge of geological events. For instance, a geological layer's geometry will differ depending on whether fault movements happened before or after the sedimentation. However, geologists usually lack the knowledge to develop visualization tools. During his Ph.D. thesis [74], Valentin Gauthier developed a geometric modeler for geologists with Jerboa. His work typically required knowledge and skills out of expertise for geologists, who might be unable to amend the modeler by adding new operations. The inference of operations could be highly valuable in that regard.

Layering

One routine operation designed and used by Valentin was a layering operation. The modeling of the subsoil usually starts from a set of surfaces. For instance, soil horizons are surfaces parallel to the soil splitting the subsoil based on intrinsic properties. The question is then how to fill the

[4] Note that usually, designing such 'hard-to-conceive' operations is also difficult because the user might not be sure of what the operation should produce.

volume between the surfaces. Indeed, several interlayers might need to be added based on stratigraphic information retrieved from core samples. In Figure 7.17, we present several possibilities for these interlayers, the Jerboa logo on the rule schemes signifies that valid rule schemes have been obtained, even though they are not displayed in full. All operations build volumes that glue two (inter)layers, i.e., two topologically equivalent surfaces. Note that the specifics of the geometry, e.g., the height of the interlayer or the volume composition, are considered business logic as it is typically within the geologist's expertise.

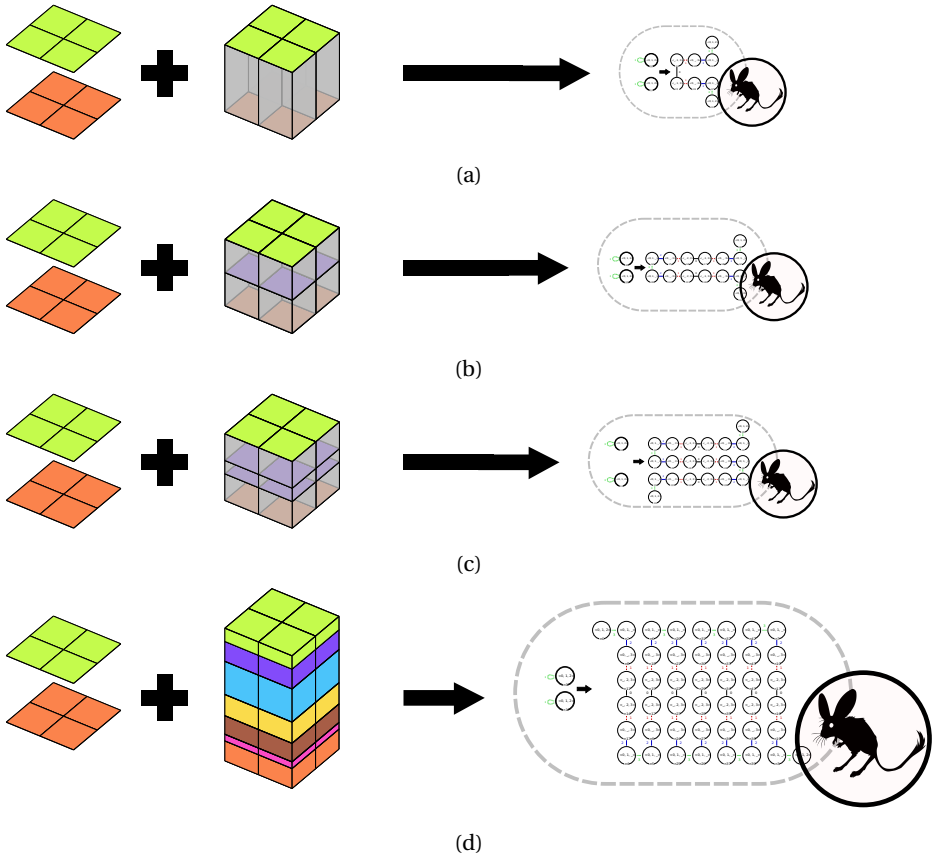
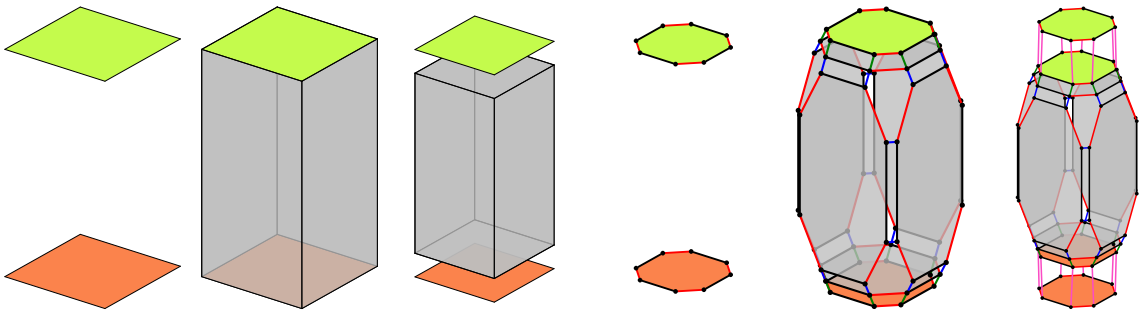


Figure 7.17: Several possibilities for the layering operation: (a) no inter-layer, (b) one inter-layer, (c) two inter-layers, and (d) six inter-layers.

We will discuss the operations of Figures 7.17b to 7.17d again after proposing a solution to deal with geometric modifications. We use the rule of Figure 7.17a to explain some constructions of the previous sections. We consider the operation as purely topological since no vertex is added. For the colors, we propagate the face colors between twin half faces and ask Jerboa to add a default color to any non-colored face (in this case, grey). We seek to build an operation that adds prisms to fill the volume between two topologically equivalent surfaces.

We first present an inconclusive attempt. Instead of providing surfaces, the user only builds two isomorphic faces, e.g., the object of Figure 7.18a. Our user builds the after instance from these two faces by adding the internal volume. For instance, they create a cube and sew the appropriate faces, resulting in the object of Figure 7.18b. The same object is represented in Figure 7.18c with its volumes exploded to help visualize that the transformation is indeed in 3D. The Gmap representations of the objects are respectively given in Figures 7.18d and 7.18e (represented in Jerboa's exploded view to ease the understanding). Then, the user builds the mapping by associating one

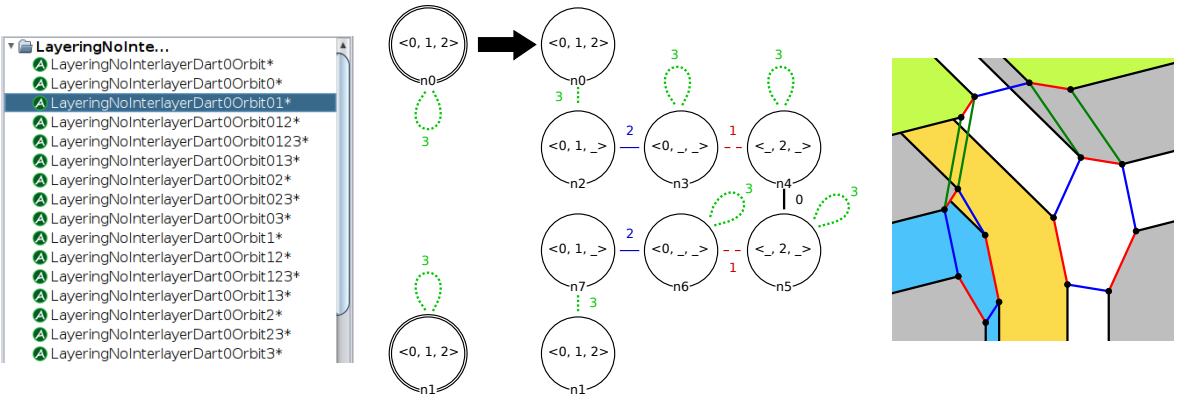
dart from each initial face to its corresponding dart in the result object (in this case, two darts), asking to map the faces based on these darts mapping. Internally, we obtain the joint representation of Figure 7.18f. Since the joint representation is connected, the algorithm will accept it as input. Our user decides not to specify the orbit type and runs the algorithm.



(a) (b) (c) (d) (e) (f)

Figure 7.18: An inconclusive attempt to infer the topological part of a layering operation: (a) initial object constituted of two faces, (b) target object describing the layering operation, (c) separation of the volumes in the target object, (d) Gmap representation of the initial object, (e) Gmap representation of the target object, and (f) joint representation or the transformation after the specification of the mapping by the user.

The user obtains the set of operations depicted in Figure 7.19a. Since our user wants an operation on a surface, they choose the operation with the orbit type $\langle 0,2 \rangle$, displayed in Figure 7.19b. Applying the inferred operation to initial surfaces with several faces will not 3-sew the volumes along the vertical faces, as depicted in Figure 7.19c. The blue and yellow half-faces (recolored for the explanation) should be 3-sewn, i.e., form a unique face. Here, we can see no such links, which, in Jerboa, means that the half-faces are 3-free (the loops are not displayed). This example highlights the consequence of choosing a non-generic enough example to infer an operation. In this case, a face is not an adequate representative example of a surface. For comparison, the desired result is shown if Figure 7.20d. Nonetheless, our algorithm found a rule, given the example provided by the user.



(a) (b) (c)

Figure 7.19: Application of the inferred operation for the inconclusive layering operation: (a) list of operations retrieved by the topological folding algorithm (one per orbit type), (b) chosen operation, the rule scheme on orbit type $\langle 0,1,2 \rangle$, and (c) zoom on a default in the application to more complex surfaces.

Now assume the same construction but with surfaces constituted of several faces, as depicted in Figures 7.20a and 7.20b. In each surface, the faces are 2-linked, and these links can be relabeled to add 3-links to glue the volumes. In the case of single-face surfaces, the 2-links were loops,

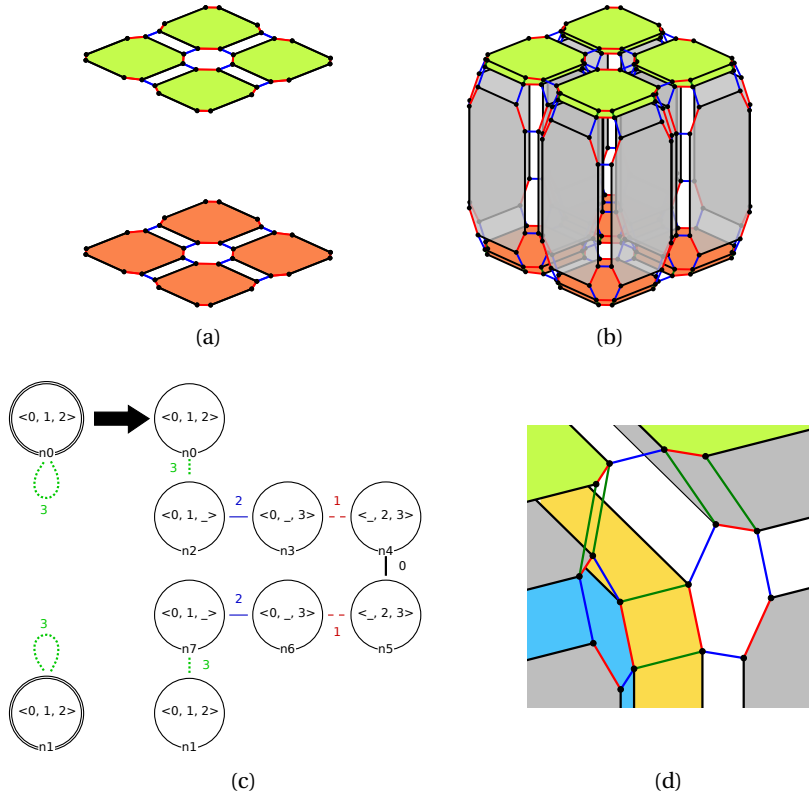
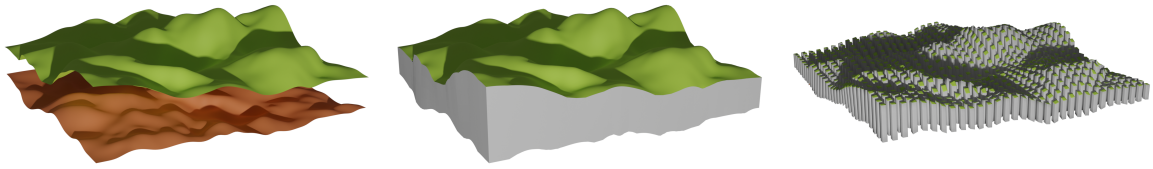


Figure 7.20: Inferring the layering operation with a representative enough example: (a) Gmap representing the initial object, (b) Gmap representing the target object (c) the inferred rule scheme on orbit type $\langle 0,1,2\rangle$, and (d) zoom on the properly 3-sewn half-faces.

meaning that the 3-loops in the object of Figure 7.20d can either be considered as the instantiation of explicit loops or as the relabeling of the 2-loops. In our implementation, the algorithm can be parametrized to map loops aggressively, meaning considering them as implicit arcs whenever possible, or conservatively, meaning always considering them as explicit loops. Note that the aggressive approach is used by default, meaning that the previous narrative can no longer happen. Empirically, we found that this approach allows for a better generalization of operations to orbits with higher dimensions. The main drawback is that topologically invalid rules might be obtained, but such rules are discarded and not proposed to the user. The parametrization between the aggressive or the conservative approach can even be realized on a dimension basis, choosing a solution separately for each dimension. Nonetheless, for some operations, some loops might come from implicit arcs and others from explicit ones (from different nodes), typically when building holes into the object. In such cases, the only viable solution is to provide an example representative enough.

The inferred operation can then be applied to more complex surfaces, as displayed in Figure 7.21. The initial surfaces are given in Figure 7.21a, the result of the operation in Figure 7.21b, and the volume explosion of the result in Figure 7.21c where the initial surfaces have been hidden to reveal the inner volumes.

Before moving on to the next operation, we want to stress that the resulting rule scheme has a disconnected left-hand side. Such a disconnected left-hand side raises no issue since the joint representation of the instantiated rule is connected. However, one hook must be specified in each connected component to obtain a valid rule scheme. Therefore, the construction of the rule



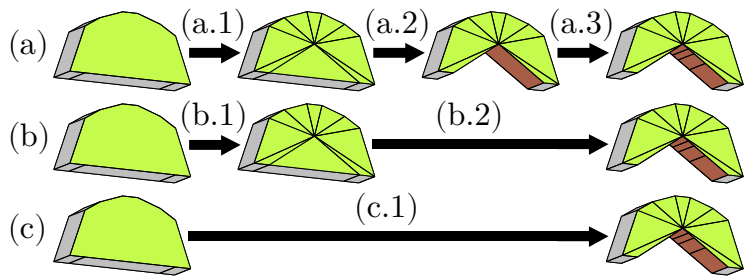
(a) (b) (c)

Figure 7.21: Application of the inferred layering operation to a complex scene: (a) initial surfaces, (b) result of the layering operation, and (c) volume explosion (after removal of the surfaces).

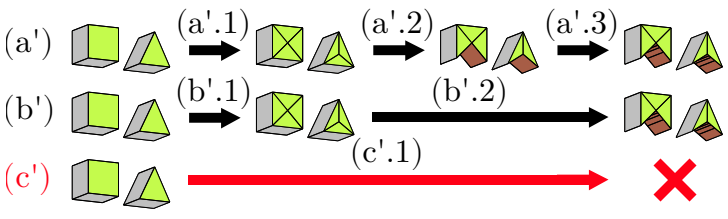
scheme looks for a full orbit (without the removing symbol '_') in each component not connected with the node already marked as a hook by the algorithm. If none can be found, the rule scheme is invalid and discarded. If several can be found, we choose a node with the same orbit type as the already existing hook, if possible, and one at random otherwise.

Natural arch erosion

Natural arches are rock formations created by geomorphological processes like erosion. A formalisation with 3-Gmaps as been proposed in [40]. The choice of 3-Gmaps was motivated by a lighter memory footprint than standard voxels and layered voxels while ensuring topologically conform objects. The model evolves via a dynamic simulation described as a set of vertex displacements evolving in time. The simulation is augmented with a collision detection loop and a set of corrective operations to solve (self-)intersections and topological inconsistencies.



(a)



(b)

Figure 7.22: Inferring a procedural generation of natural arches: (a) three procedural generations producing the same target object with different numbers of intermediate steps, and (a) applications of the generation from other prisms, the third generation is not applicable.

We used our inference mechanism to develop a procedural approach to obtain similar formations as 2D objects. In Figure 7.22a line (a), we distinguish three operations in the generation: step (a.1) triangulates both bases of the initial prism; step (a.2) removes the bottom part of the prism, adding quad faces to close the surface; step (a.3) splits the faces added in the previous step. In Jerboa, step (a.1) is described with an orbit-type $\langle 0,1 \rangle$ while steps (a.2) and (a.3) correspond

to rules with an orbit-type $\langle 0,2\rangle$. Therefore, we can optimize the set of rules and directly infer the operation (b.2) that removes the volume and splits the bottom faces at once. We obtain the two-rule procedural generation (b). One might want to optimize the generation again and obtain the approach (c) that only contains one rule. However, this rule cannot be parameterized by any meaningful orbit type resulting in the impossibility of applying it to other prisms. We still obtain a procedural construction, but it would be only applicable to the prism used for inference. For instance, we can apply the inferred operations to a triangular prism or a cube, as depicted in Figure 7.22b. Procedural generations (a) and (b) are still valid, but generation (c) is not, as signified by the red cross. These rule sets illustrate two aspects of our inference mechanism. First, we can see the importance of the orbit type used in the inference. The orbit types define a lattice with respect to the inclusion of dimension sets, and the least upper bound of two orbits provides a more generic operation (provided it exists). Secondly, the rule inference mechanism can be used as a routine to decrease the number of rules in a procedural generation by inferring the result of a sequence of operations. However, an aggressive approach, decreasing the number of rules too drastically, may produce specialized operations as in generation (c). Such an approach results in a loss of expressiveness compared to the initial procedural modeling. The user should choose the appropriate trade-off.

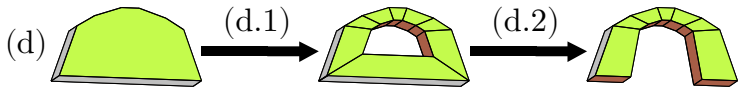


Figure 7.23: Inferring another procedural generation of arches.

Other arch generations are possible. For instance, the arches in the cover figures of Chapters 3 and 4 have been generated from the procedural construction of Figure 7.23. It suffices to alter the representative example used and rerun the algorithm to obtain variations of a common theme. Using the prisms, we obtain the arched of Figure 7.24.

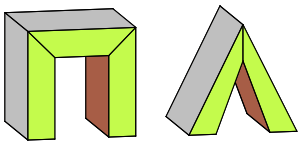


Figure 7.24: Results for the procedural generation inferred in Figure 7.23.

7.4.2 Application to subdivision schemes for surface refinement

Subdivision schemes for surfaces allow recursively refining a coarse mesh into a finer one. In computer graphics, they are used to estimate a smooth limit surface by either approximation or interpolation. Similar refinement techniques are used in numerical methods for solving partial differential equations, e.g., in the finite element method. We propose to reconstruct subdivision schemes for surfaces, i.e., with the orbit type $\langle 0,1,2\rangle$ as the inference parameter. In this section, we will mostly discuss topological reconstructions. The missing geometric expressions may be found on the Jerboa's webpage[5] with the demo[6] of our publication [138].

[5] Link to website (last consulted on September 24th, 2022): http://xlim-sic.labo.univ-poitiers.fr/jerboa/doc/topological-inference-for-subdivision-schemes/.

[6] Link to the demo (last consulted on September 24th, 2022): https://youtu.be/hp1QK1ZRrKk.

Quad subdivision and Catmull-Clark

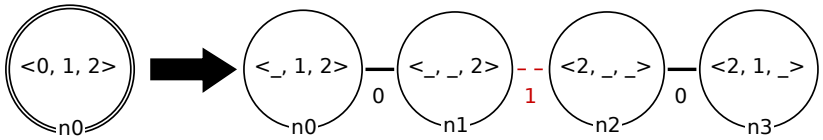


Figure 7.25: Inferred rule scheme for the quad subdivision of surfaces from the objects of Figures 7.2.

We can use the folding algorithm to infer the rule scheme for the quad subdivision, which has already been discussed extensively throughout this dissertation. Using the Gmaps built from the objects of Figure 7.2a and the orbit $\langle 0,1,2\rangle$, we obtain the rule scheme given in Figure 7.25. This rule scheme corresponds to the one presented in Chapter 6 made manually with Jerboa's rule editor. We add embedding expressions computing the position of the new vertices as the barycenter of the faces and edges, which yields a complete operation. This operation was used in the iterative sequence of Figure 7.26a to 7.26d. Since the quad subdivision is topologically equivalent to the Catmull-Clark subdivision [31], we can tweak the inferred operation with an embedding expression smoothing the surface. Therefore, the rule scheme of Figure 7.25 also describes the Catmull-Clark subdivision. The only difference concerns the geometric computations added. For instance, the vertex added to split the edge corresponds to the rule's nodes $n1$ and $n2$. For the quad subdivision, the new position is computed as the middle of the edge's vertices with the expression:

```
// midpoint of the edge
return Point3::middle(<0>_position(n0));
```

The smoothing for the Catmull-Clark subdivision requires translating the original vertex (identified by n0). Using the standard geometric refinement from [31], we obtain the expression:

```
// midpoint of the incident face
Point3 face1Mid = Point3::middle(<0,1>_position(n0));
// midpoint of the adjacent face
Point3 face2Mid = Point3::middle(<0,1>_position(n0@2));
// average of the face points
Point3 faceMid = Point3::middle(face1Mid, face2Mid);
// midpoint of the edge
Point3 edgeMid = Point3::middle(<0>_position(n0));
// average of the edge and face points
return Point3::middle(faceMid, edgeMid);
```

The smoothing computation on the inferred operation yields the desired refinement scheme, illustrated in Figures 7.26e to 7.26h.

Loop and Butterfly

One standard refinement of triangulated surfaces is the Loop subdivision scheme [124]. This scheme splits each edge by inserting its midpoint vertex. The new vertices are linked with edges in each face, dividing each triangle into four new triangles. One iteration on a triangle is illustrated in Figures 7.27a and 7.27b. From these two instances, we infer the rule scheme of Figure 7.27c. We added the standard masks to compute the position of the new vertices and Warren's simplified weights [180] for smoothing the old vertices. This subdivision approximates the initial surface.

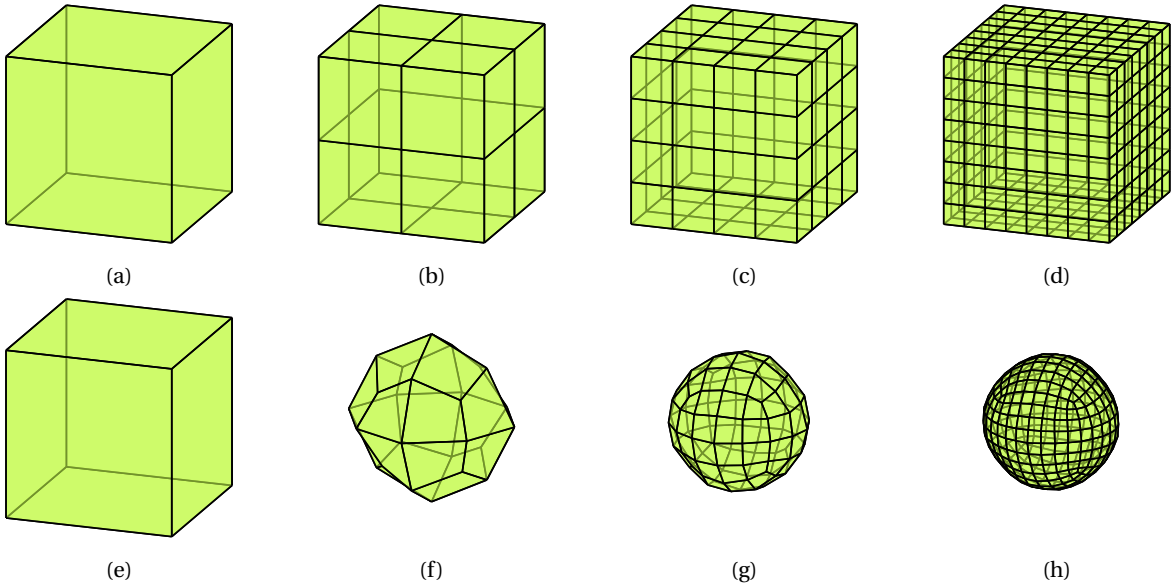


Figure 7.26: The quad subdivision operation is topologically equivalent to Catmull-Clark: iterative sequence for the quad subdivision (a), (b), (c), and (d); iterative sequence for Catmull-Clark (e), (f), (g), and (h).

Choosing different geometric computations yields the interpolation scheme known as the butterfly subdivision [51]. Iterations of the subdivisions are provided in Figure 7.28.

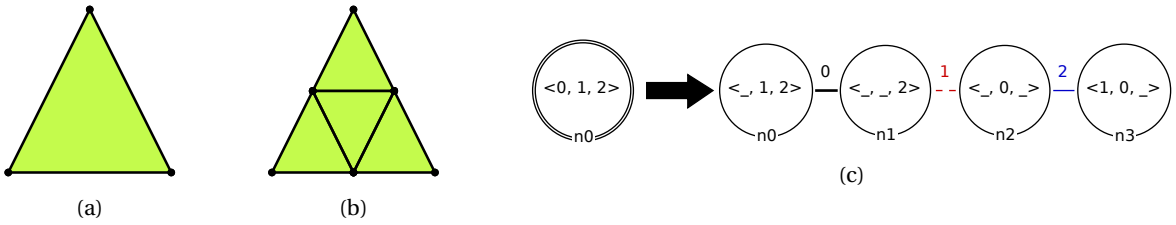


Figure 7.27: Inferring the Loop subdivision: (a) the initial object, (b) the first iterations of the subdivision, and (c) the inferred operation.

Powell-Sabin

Powell and Sabin studied various subdivision schemes on triangles [146] that ensure given values for the first derivatives on vertices. Such subdivisions are still studied to construct smooth finite element spaces [85]. The Powell-Sabin 6-split refines a triangle into six new triangles. The operation adds a vertex at the center of the face, links it with all vertices of the triangle, and finally inserts all midpoints of the initial edges. Assuming we only have three basic operations: split an edge, add a vertex, and link two vertices by an edge, we can reconstruct the Powell-Sabin 6-split refinement as described in Figure 7.29a to 7.29d. Inferring the rule scheme of Figure 7.29h, for the connected component (orbit $\langle 0,1,2\rangle$), takes around 5ms. We apply the inferred operation on the triangulation of the unit square illustrated in [85], see Figures 7.29e to 7.29g. Conversely to other subdivision schemes, the Powell-Sabin 6-split refinement was not already implemented in Jerboa. However, we obtain visually similar results on the unit square as in [85].

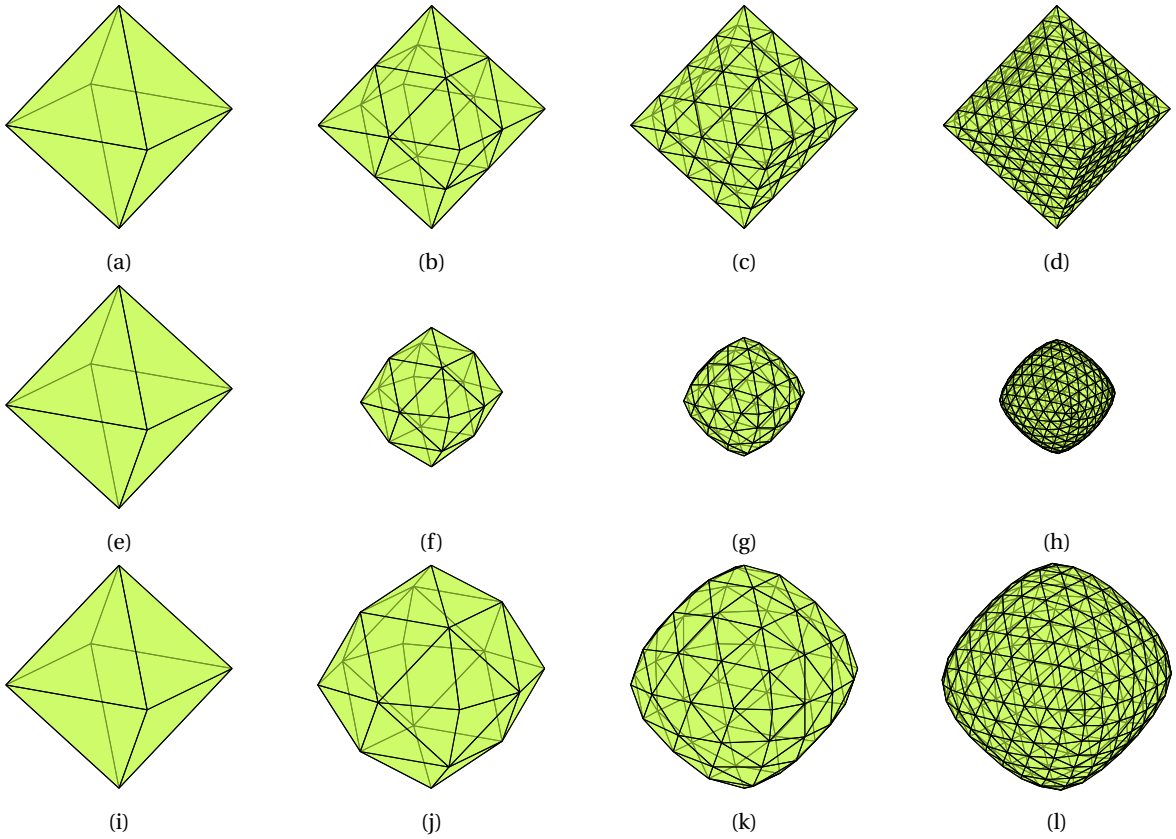


Figure 7.28: Triangular subdivisions with topologically equivalent refinements: iterative sequence for the non-smoothed subdivision (a), (b), (c), and (d); iterative sequence for Loop algorithm (e), (f), (g), and (h); iterative sequence for Butterfly subdivision (i), (j), (k), and (l).

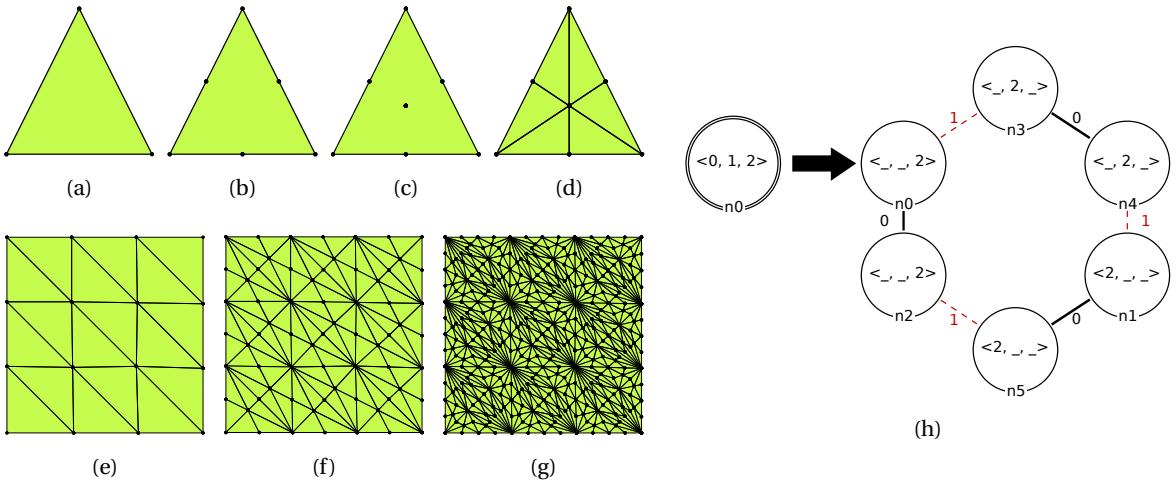
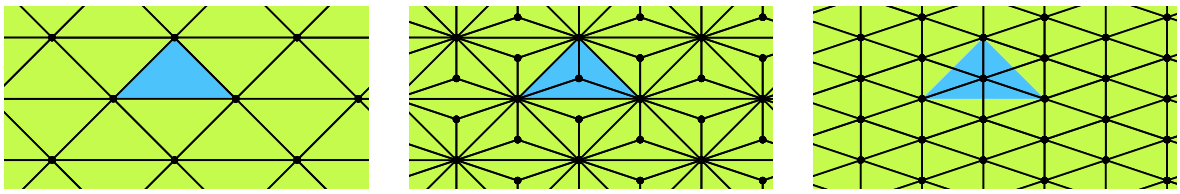


Figure 7.29: The Powell-Sabin 6-split refinement: given a triangle (a), split the three edges (b), add a vertex (c), and link it with all other vertices (d); the inferred rule (h) is recursively applied on a triangulation of the unit square (e) (from [85]) to obtain more refined triangulations (f) and (g).

Kobbelt's $\sqrt{3}$

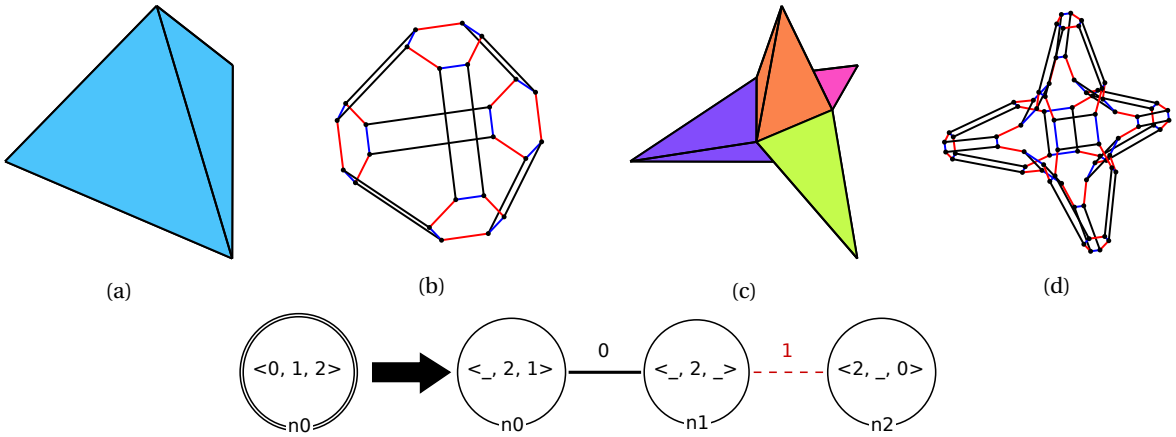
The $\sqrt{3}$-algorithm developed by Kobbelt [110] is topologically distinct from the previous subdivision schemes in the sense that it removes (or rather flips) edges. The algorithm is usually described in a two-step process. First, vertices are added at the center of each triangle and linked with edges to the original face vertices. From the triangles of Figure 7.30a, we obtain the mesh of

Figure 7.30b. One initial triangle is colored blue to help visualize the transformation. Then, every edge of the initial mesh is flipped, as illustrated in Figure 7.30c. We can obtain the transformation as a single-step operation, similar to the construction of the natural arches in Section 7.4.1. We inferred this operation on a tetrahedron. The initial and final objects are provided in Figures 7.31a and 7.31c. The associated 2-Gmaps are given in Figures 7.31b and 7.31d, while the inferred operation is illustrated in Figure 7.31e. With the addition of the vertices' geometric smoothing, we obtain the subdivision scheme defined by Kobbelt. An illustration of the final operation on the Stanford Bunny is provided in Figure 7.32, with one initial triangle highlighted in blue.



(a) (b) (c)

Figure 7.30: Two-step illustration of the $\sqrt{3}$ algorithm: from an initial triangular mesh (a), faces are triangulated with a vertex at the center (b), and old edges are flipped (c).



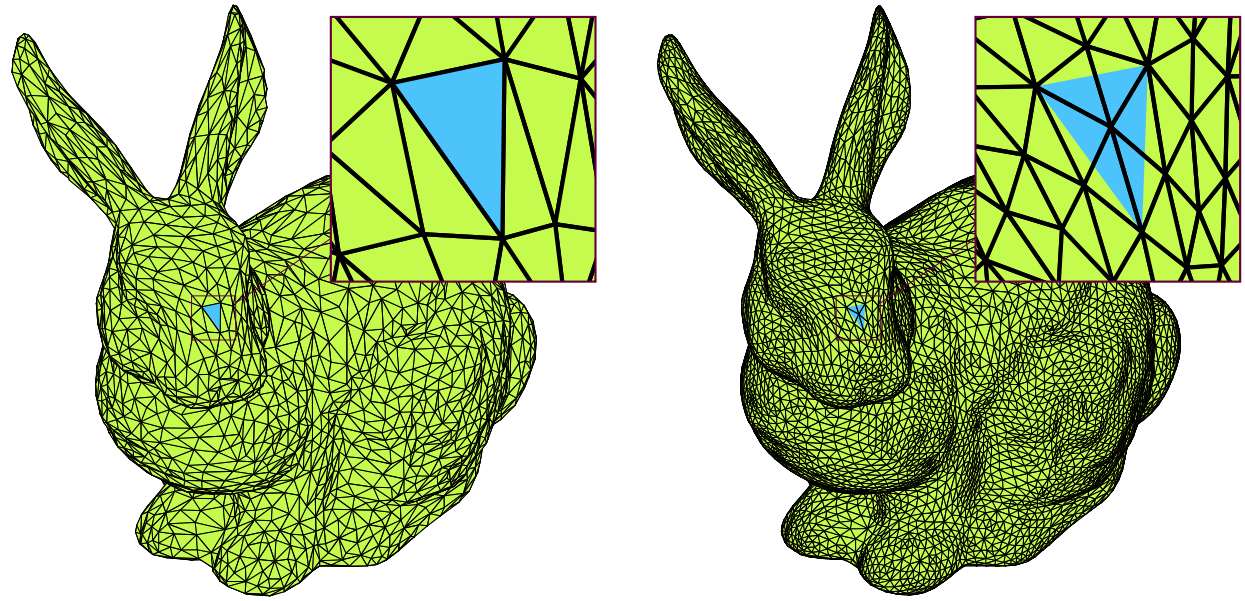
(a) (b) (c) (d)

(e)

Figure 7.31: Inferring the $\sqrt{3}$ operation: (a) tetrahedron as the "before" instance, (b) Gmap of the "before" instance, (c) refined object as the "after" instance, (d) Gmap of the "after" instance, and (e) inferred rule scheme for the orbit $\langle 0,1,2\rangle$.

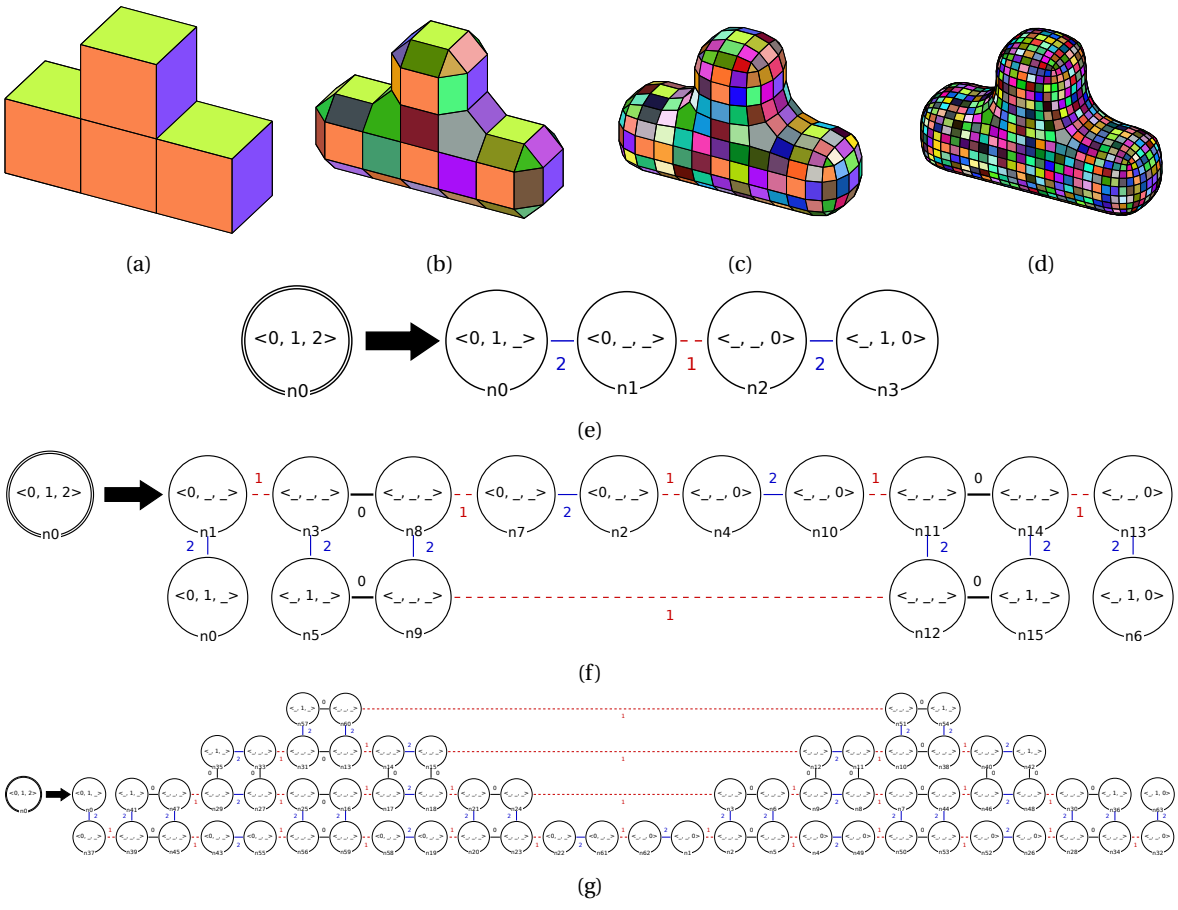
Doo-Sabin

The Doo-Sabin subdivision works with any surface [48] by recursively splitting the vertices. Figure 7.33a presents the initial object, while Figure 7.33b displays the first iteration of the subdivision. From these two objects, we infer the rule of Figure 7.33e. We now have a standalone rule applicable to isolated surfaces, which refines objects as precisely as desired. For instance, we can further subdivide the object of Figure 7.33b to obtain the second and third iterations, respectively illustrated in Figures 7.33c, and 7.33d. From these iterated applications, we can infer new operations directly producing the second or third iteration of the Doo-Sabin subdivision. The inferred rules are illustrated in Figures 7.33f and 7.33g. In other words, our mechanism allows for straightforward self-composition of operations. Inferring the rule of the third iteration takes around 40ms, which means our approach is usable in practice.



(a) (b)

Figure 7.32: Applying the inferred operation for the $\sqrt{3}$ subdivision, after the addition of the missing geometry: (a) initial object with 4968 faces, (b) result of one subdivision (object with 14904 faces).



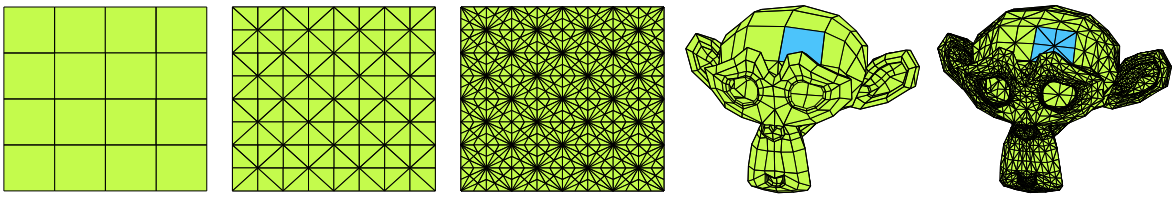
(a) (b) (c) (d)

(e)

(f)

(g)

Figure 7.33: Inferring the Doo-Sabin subdivision: (a) an initial object, (b) the first iteration of the subdivision scheme, (e) the inferred operation from the objects (a) and (b), (c) and (d) the second and third iterations of the subdivision scheme, (f) inferred operation for the second iteration from the objects (a) and (c), and (g) inferred operation for the third iteration from the objects (a) and (d).



(a) (b) (c) (d) (e)

Figure 7.34: Powell-Sabin 6-split on quad meshes: (a) a quad subdivision of the unit square, (b) its triangulation with Powell-Sabin 6-split, (c) further refinement of the triangulation, (d) a quad subdivision of Suzanne, and (e) its triangulation with Powell-Sabin 6-split.

7.4.3 Advanced exploitation: target orbit type parameter

We now present an atypical consequence of operations inference in our dedicated model. Operations are inferred for a given orbit type, i.e., an abstraction of the topology. For instance, we inferred the Powell-Sabin 6-split operation with the orbit type $\langle 0,1,2\rangle$, specifying that the operation modifies a surface. We never specified that the operation subdivides triangles. Therefore, we can apply it to any surface. For instance, we can use this operation to triangulate the quad mesh of Figure 7.34a. We obtain the triangulated mesh of Figure 7.34b. We can iterate the subdivision to obtain the mesh of Figure 7.34c. Similarly, the representation of Suzanne in Figure 7.34d is a surface that only consists of quads. We can use the inferred operation of Figure 7.29h to obtain a mesh triangulation, as illustrated in Figure 7.34e.

The proposed algorithm folds the graph correctly to obtain a rule by traversing all darts of the application example. Therefore, we obtain an unconventional way of designing new operations. First, based on Jerboa's language, we obtain operations parameterized by an orbit type independent of the underlying topology. For instance, the object no longer corresponds to a triangulation or a quadrangulation but simply to a surface identified by the orbit type parameter of the operation. Secondly, we can obtain new operations by modifying the orbit parameter of the topological folding algorithm. For instance, we can generalize some operations from faces to surfaces and even volumes. The generalized operations also come with the guarantee that their application to a well-formed Gmap produces a well-formed Gmap.

7.5 Algorithm analysis

In this last section, we provide an overview of the analysis of the topological folding algorithm. The complete proofs of complete correctness and complexity can be consulted in Appendix C.

7.5.1 Correctness analysis

The proof of correctness essentially results from one fact. At each step of the algorithm, the partial graph pattern correctly folds all links incident to darts associated with a node with a defined orbit type and expanded arcs. The cases of failures for the algorithm are either the impossibility of adding an arc to the partial graph pattern or the impossibility of determining a legal relabeling function to add to a node. In the case of a node failure, the issue translates into an asymmetry in operation. Conversely, an arc failure means a discontinuity in the topology. The issue on an arc is actually twofold. Either all other endpoints of the links do not share the same association with

a node in the graph scheme (instantiation failure), or the other endpoints do not correspond to the image of the same dart from the orbit used as a reference for the instantiation (arc failure). The issue related to the node's orbit type is relatively less complicated, as the only real issue is the hopelessness of building a joint relabeling value for a given dimension.

7.5.2 Complexity analysis

The topological folding algorithm is an enhanced graph traversal algorithm. Similar to standard graph traversal algorithms, the algorithm is linear in the size of the graph (the number of nodes plus the number of edges). The proof results from the fact that each dart can only be considered a bounded number of times: once from each neighbor. In our case, a dart has a most one neighbor for each dimension in $0..n$ and one κ-neighbor. In particular, the complexity does not depend on the size of the orbit type or the orbit graph.

Summary of the chapter's contributions

We presented an automated method to infer the topological part of modeling operations from their description on a representative example. Our approach is highly facilitated by the regularity of the underlying model, namely Gmaps, and the genericity of rule schemes in Jerboa as a solution to express modeling operations. The topological folding algorithm takes as input an instance of the operation application (two Gmaps and a partial mapping between them) and the orbit type parameter. From these inputs, the algorithm essentially tries to undo the process of Jerboa's rule instantiation by folding a joint representation of the transformation. It outputs a rule scheme with a hook decorated with the provided orbit type. We showed the correctness of the method: if the algorithm produces a rule scheme, then it instantiates as the initial transformation; if the algorithm produces no such rule scheme, then none exists. We experimented with our algorithm on various objects with applications for geology and subdivision schemes. The inferred operation is directly applicable to any object by matching the rule's hook into an orbit of the appropriate type in the object. Our approach offers the following three main benefits:

- A user unaccustomed to either Gmaps or (graph) transformation rules can design operations exclusively from examples.
- A sequence of elementary operations can be optimized to generate a direct transformation that can speed up the design of complex scenes.
- An operation can be generalized granted some regularities in the modification: first, from a specific topology, e.g., triangles, to any orbit of the same type; secondly, from an orbit type to a more general one, e.g., from a face to a surface.

Our approach ensures that the topological part of modeling operations can be inferred without writing a single line of code in a standard programming language or designing a rule in Jerboa's expert language. It also hides the intern structure of Jerboa's rules. Finally, an automated and reliable inference mechanism offers an alternative approach to the development of topology-based geometric operations. Non-expert users can thereby develop their own operations.

The inferred rules were manually edited to add the missing geometric computations before applying them again. We will see in the next chapter how to retrieve plausible embedding expressions to add to the nodes, providing a complete description of modeling operations.

Chapter 8

Inference of embedding expressions

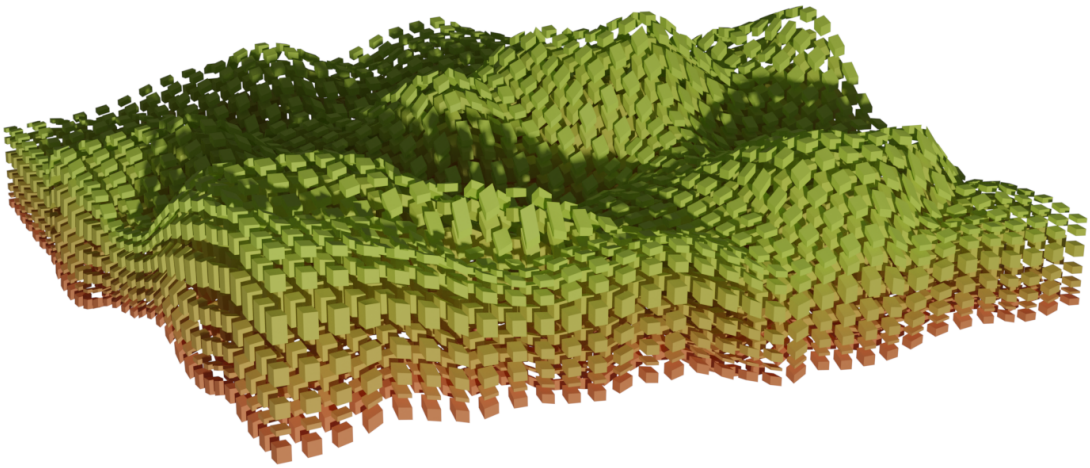


Figure 8.1: In Chapter 7, we inferred the topological part of a layering operation. Now, how can we retrieve the embedding expressions that automatically compute the geometric modifications?

Personal note on the chapter

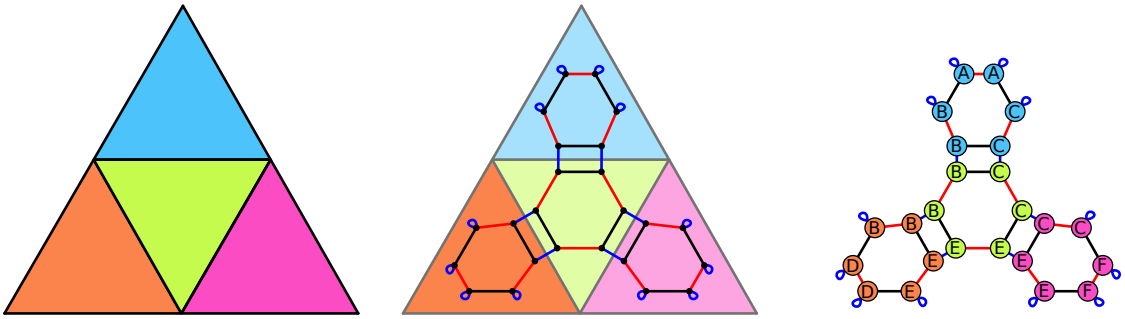
Inferring the topological part of geometric modeling operations exploited the formalism of graph transformation rules and the abstraction of the topology via rule schemes. We proposed a deterministic approach and showed that it could effectively be used to reconstruct topological modifications.

In Jeboa's rule language, the geometric modifications are computed via embedding terms. These terms offer unbounded freedom in the possible computations. In particular, we can implement complex computations in a programming language and call the computation from Jerboa. In particular, it would be delusive to assume that we can retrieve any computation. However, we will see that if we delimit the scope of solutions, methods based on optimization and operational research allow retrieving the computations. In this chapter, we illustrate the methodology using linear functions.

Contents

In Chapter 7, we presented the topological folding algorithm as a solution to infer the topological part of a geometric modeling example. In Section 7.4, we showed some results of our algorithm. However, the inferred rules were manually completed with embedding expressions such that we could apply them and visualize the results. This chapter presents a method to infer these embedding expressions, still assuming a representative example. More precisely, the method proposed in this chapter exploits information obtained during the topological folding to guide the reconstruction of the embedding expressions. Together with the topological inference, we obtain a solution to deduce operations entirely.

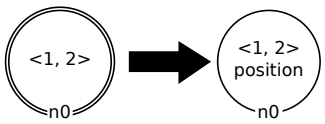
8.1 Geometric modifications in rule schemes



(a) (b) (c)

Figure 8.2: Embedded representation of four triangles: (a) geometric object, (b) the topological representation displayed over the faded object, and (c) the embedded representation of the object, each dart stores a color value and a position value, depicted directly on the darts.

We use the embedding representation given in Chapter 5 for the construction of embedded Gmaps. For instance, the geometric object of Figure 8.2a admits the topological representation of Figure 8.2b, while the embedded Gmap is given in Figure 8.2b.



(a)

```
// Position expression for n0
Point3 p = new Point3(n0.position);
p.addVect(vect);
return p;
```

(b)

Figure 8.3: Rule scheme for the vertex translation: (a) rule schemes, and (b) embedding computations attached to node $n0$.

In this chapter, we will stick to the representation of rule schemes as obtained in Jerboa's rule editor since the goal is to reconstruct them. For instance, the operation for the translation of a vertex is given in Figure 8.3. The rule scheme in Figure 8.3a modifies an orbit $\langle 1,2 \rangle$, i.e., a vertex. Topologically, no modification occurs since node $n0$ is decorated with the same orbit type in both sides of the rule. However, the `position` keyword on node $n0$ in the right side indicated a geometry modification. The associated embedding computation is given in Figure 8.3b. The new point is created and initialized with the position value of $n0$, to which we add a vector `vect` before returning the value. This computation corresponds to the expression $\texttt{plus}(pos(x), \vec{v})$ used in Chapter 5 for the vertex translation.

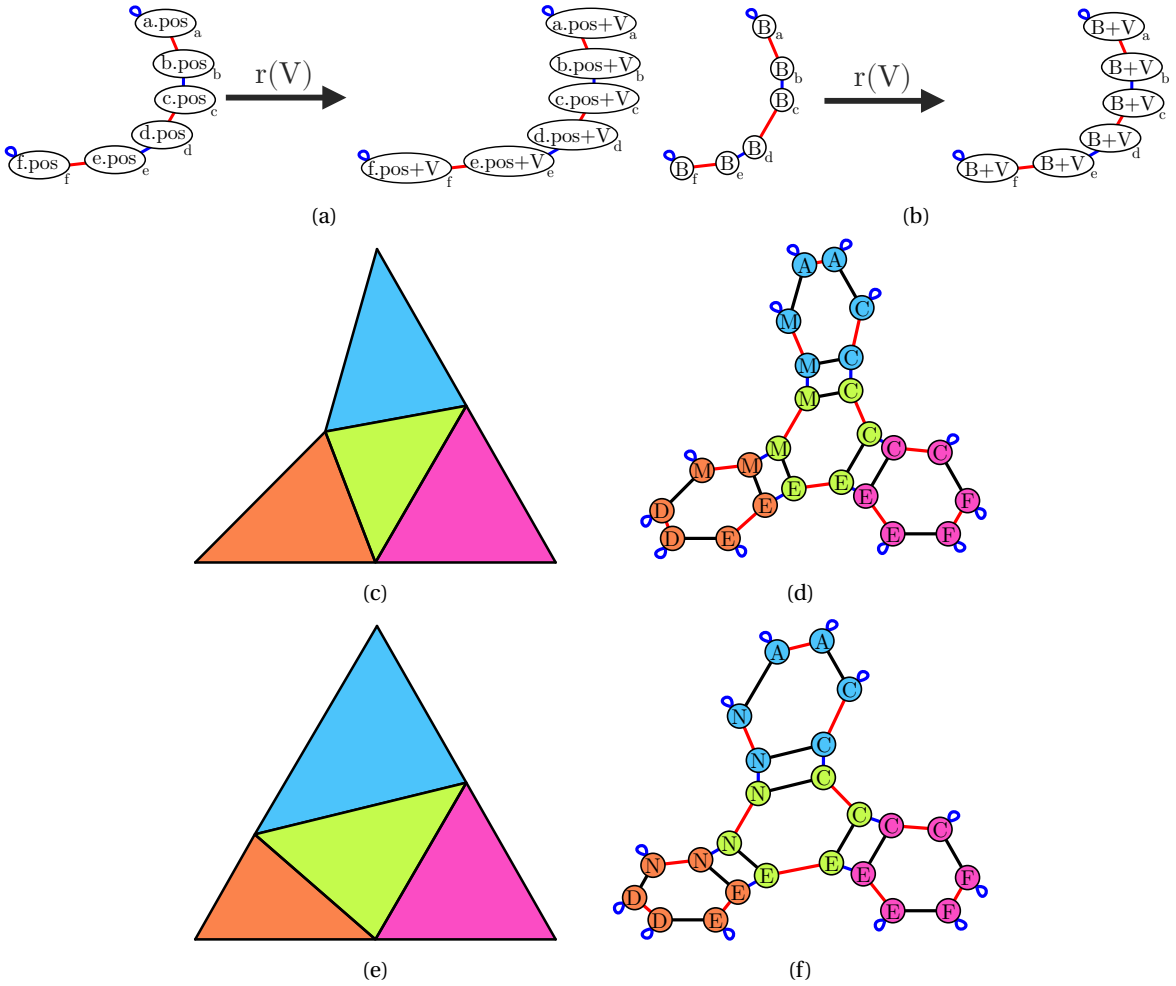


Figure 8.4: Embedding expressions for the vertex translation: (a) the topological instantiation of the rule scheme for the vertex translation, (b) associated embedding modification, (c) result of the application with a specific value for V, i.e., when $B + \vec{V} = M$, (d) the associated Gmap, (e) result of the application with a different value for V, i.e., when $B + \vec{V} = N$, and (f) the associated Gmap.

Mapping the hook $n0$ (Figure 8.3a) to any dart of the vertex embedded with the position B (Figure 8.2c yields the rule of Figure 8.4a. In this instantiated rule, $n0$ is substituted in the expression by the identifier of the associated nodes (a to f), as described in Section 6.3.3. The actual computation replaces the term $a.pos$ to $f.pos$ with B, and we intuitively obtain the computation described in Figure 8.4b. Different values of V, i.e., values of `vect` in Figure 8.3b, yield different vertex translations. For instance, both objects of Figures 8.4c and 8.4e can be derived. We provide the embedded Gmaps of these objects respectively in Figures 8.4d and 8.4f.

In Chapter 7, we explained how to retrieve the topological part of the rule scheme, i.e., the content of Figure 8.3a, from a partial mapping of a before instance to an after instance, e.g., between the graphs of Figures 8.2c and 8.4d. This chapter aims to extend the inference mechanism to deduce the embedding expression of Figure 8.3b.

8.2 Inferring position expressions on rule schemes

We start with the position embedding, which is crucial to display objects. Rather than studying the inference of the embedding expressions as a standalone problem, we propose to solve it based

on the information retrieved from the topological folding algorithm. Therefore, the input of the method is extended with the following:

- a rule scheme obtained from the topological folding algorithm (Algorithm 6 in Chapter 7),
- the association between the node in the rule scheme and the darts in the instances,

We start with the simplest case, the result of the topological folding algorithm on the empty orbit type $\langle\rangle$. The aim is to introduce hypotheses on the problem and some tools we will use afterward in the complete framework.

8.2.1 Case of the empty orbit type

Recall from Chapter 7 that the topological algorithm (Algorithm 6) always succeeds with the empty orbit since no topological abstraction is realized and the obtained rule only encodes the mapping given by the user. The goal is now to retrieve the embedding expressions attached to the nodes in the right-hand side.

In the context of a rule scheme on the empty orbit, the input of the method consists of the following:

- an object before modification and its counterpart part after the operation, represented as embedded G-maps,
- the match to the before G-map and the comatch to the after instance,
- the mapping from the occurrence of the match to the occurrence of the comatch.

We illustrate all these notions with Figure 8.5. The operation we seek to retrieve is a barycentric triangulation of a square: a new vertex is added at the middle of the square and linked with an edge to the four initial vertices of the square. Figures 8.5a and 8.5b respectively display the initial object and the target one. The goal is to add the missing embedding expressions on the rule at the bottom of Figure 8.5c. In Figure 8.5c, the match corresponds to the dashed grey links in the left part of the Figure, while the comatch to the dashed grey links in the right part of the Figure. Finally, the mapping describing the rule is described with pink links.

We can access the values of each right node in the after instance through the co-match. We call it the after value. Via this after value, we can check that an embedding expression added on the right-hand side is correct, i.e., yields the correct value. We can also access its embedding value in the before Gmap, provided that a counterpart left node exists. Indeed, the value can be obtained by finding the counterpart left node and then the corresponding node in the before instance through the match. We call this value the before value. Note that this construction is symmetric, and we can access values for the left nodes in the before and after instances. We will also call these values before and after values.

The most naive solution is to build the embedding expression of each right node n_r as follows:

1. If n_r is a preserved node whose after value coincides with its before value, then no computation needs to be realized, and we leave the expression of the right node empty.
2. Otherwise, if its after value coincides with the before value of a left node n_l, then assigning the value yields the wanted result, and we add the expression $n_l.pos$ to n_r.

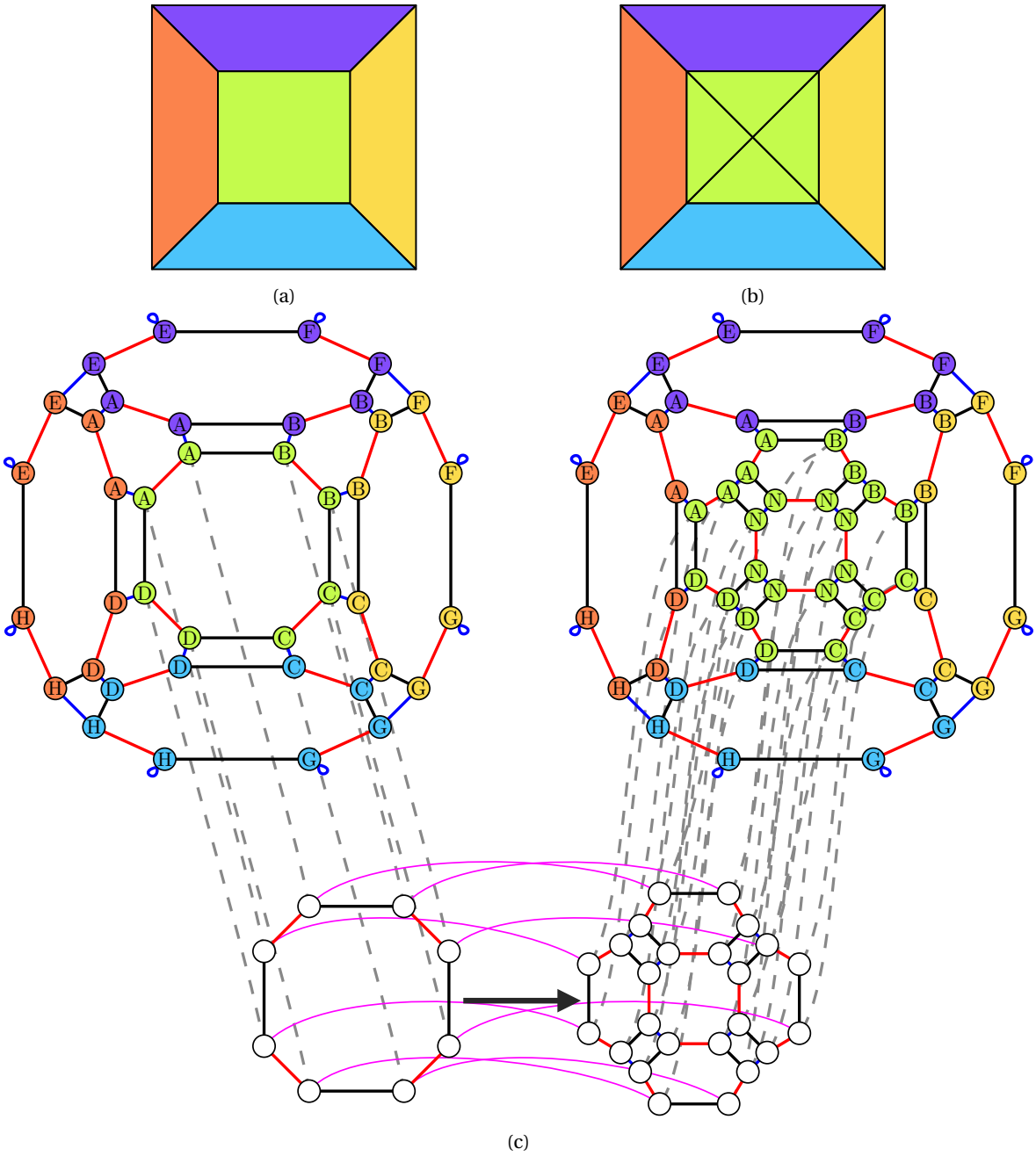


Figure 8.5: Context for the inference of embedding expressions: (a) initial object, (b) target object, (c) topological rule with match and co-match (grey dotted lines) and partial mapping (pink lines).

3. If none of the above cases is possible, we decide that the new value corresponds to a global variable of the rule X and add the expression X to n_r.

If several right nodes share the same after value which leads to using a global variable, we first check whether the after value already corresponds to a global variable before creating a new one. Applying this solution yields the rule of Figure 8.6. The positions of A, B, C and D do not change, so the preserved nodes (a, b, h, i, p, q, w, and x) do not have an expression and the expressions for the added nodes (c, f, g, j, o, r, v, and s) exploit accessors from the left nodes. Finally, the position of the middle vertex leads to the addition of a global variable N used for assigning values to the nodes (e, d, l, k, m, n, u, and t).

However, the purpose of inferring an operation is to deduce as much information as possible.

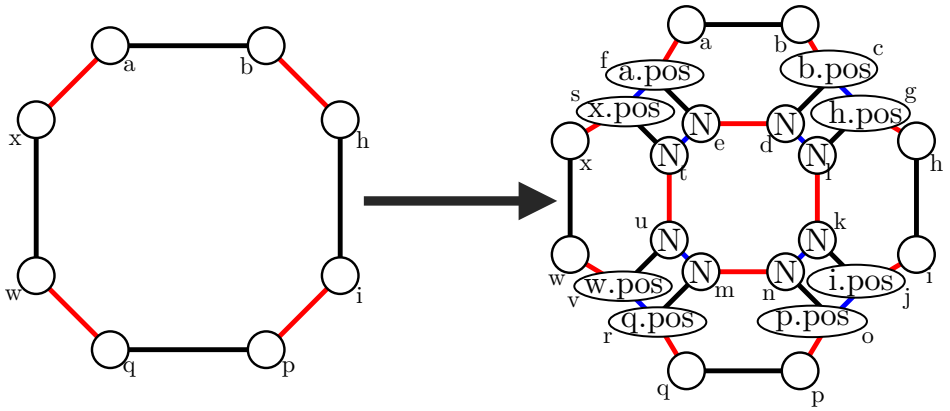


Figure 8.6: Embedded rule obtained by naive inference for the triangulation of a face, only accessors and global variables are allowed.

For instance, we might want to reconstruct the position N instead of turning it into a global variable. If we assume a barycentric triangulation, the position N in Figure 8.5c can be computed as the barycenter of A, B, C, and D:

$$\mathrm{N} = \frac{1}{4}(\mathrm{A} + \mathrm{B} + \mathrm{C} + \mathrm{D})$$

In such a case, the position of the points can be retrieved with the accessors and N could be replaced by $\frac{1}{4}(a.pos + b.pos + i.pos + q.pos)$ in the rule of Figure 8.6. The question we ought to solve is how to retrieve such an expression.

Since we know the before and after values, we propose to find a computation of the after values based on the before value. More precisely, we modify case 3. Instead of deciding that the new value corresponds to a global variable, we assume that the new value can be computed as a function of the before values. In the case of the triangulation, we are looking for a function f such that $\mathrm{N} = f(\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D})$. First, we need to specify the class of functions that we are looking for. In this document, we will look for functions expressed as linear combinations of their arguments. For instance, we would look for:

$$f(x_1, x_2, x_3, x_4) = w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3 + w_4 \times x_4$$

where w_1, w_2, w_3, and w_4 are weights. We can evaluate this function thanks to the before and after values to obtain the equation:

$$\mathrm{N} = w_1 \times \mathrm{A} + w_2 \times \mathrm{B} + w_3 \times \mathrm{C} + w_4 \times \mathrm{D}.$$

Therefore, determining the expression of the function can be reduced to solving a linear equation. An example of solution is $w_1 = w_2 = w_3 = w_4 = \frac{1}{4}$. More generally, for a node n_r in the rule's right-hand side, we are searching for a solution to the equation

$$pos(n_r) = \sum_{n_l \in \mathrm{V_L}} w_{n_l} pos(n_l) + t \tag{8.1}$$

where $\mathrm{V_L}$ is the set of nodes of the rule's left-hand side, $(w_{n_l})_{n_l \in \mathrm{V_L}}$ are (unknown) weights, and t encodes an (unknown) intrinsic translation. Therefore, we consider functions encoded as affine combinations of the initial points. We call *family of functions* the set of functions that we consider

for solving the geometric inference, i.e., the search space.

The linearity hypothesis is made such that solutions can effectively be found. One can extend the study to polynomials, trigonometric polynomials, or any desirable family of functions. The hypothesis essentially depends on the kind of solutions that we are expecting to find and, from a practical point of view, the ease of solving the equation. We model the equation as a constraint satisfaction problem (or CSP). A CSP is defined as a set of variables with a domain of values for each variable and a set of constraints on the variables. In our case, the variables are the weights with domain $\mathbb{R}$, and the only constraint is equation 8.1. A solution of a CSP is an evaluation of the variables that satisfies all the constraints. Once a problem is modeled as a CSP, we can delegate its resolution to dedicated solvers. We tried two solvers OR-tools from Google [78] and Z3 from Microsoft [131] but mostly worked with OR-tools, which tends to produce simpler solutions. For our purposes, these solvers can be viewed as black boxes to which we feed equations and from which we get a solution (or the information that no such solution exists). In the next section, we discuss how to extend this approach to rule schemes.

8.2.2 General case

In general, the output of the topological folding algorithm is a rule scheme parameterized by a non-empty orbit type. From the explanation in Section 8.1 and Chapter 6, we know that the node identifiers in the embedding expressions of the rule scheme will be substituted by dart identifiers from the occurrence of the match. Thus, the main difficulty is to infer an expression on a node that is valid for all darts images of the node by the instantiation.

Intuitively, we can extend the context of Figure 8.5c to that of Figure 8.7. The inferred rule scheme with the orbit type $\langle 0, 1 \rangle$ is depicted at the bottom. Since the rule scheme was obtained via the topological folding algorithm, we can exploit the association between the node in the rule scheme and the darts in the before and after instances. These associations are represented via the colors on the nodes. For instance, node $n0$ is associated with the 8 blue nodes in both the left-hand and right-hand sides of the mapping. Similarly, node $n1$ is associated with the 8 orange nodes, and node $n2$ with the 8 purple nodes. Each node ($n0$, $n1$, and $n2$) now encode the modification of all their associated dart. Since the end goal is the rule scheme, we can no longer perform computation on a dart basis.

Intuitively, the issue comes from the topological abstraction inherent to rule schemes. However, embedding expressions offers solutions to match this topological abstraction. As seen in Chapter 5, the collect operators allow writing computation based on multisets of values computed from topological entities.

8.2.3 Points of interest

The statement is simple: we need to replace the positions of the darts with values that can be computed from expressions on nodes while encoding abstraction of the topology. We call *point of interest* any such solution.

Similar to the choice of linearity hypothesis for the family of functions, we need a choice for computing points of interest. We propose to use barycenters of orbits. For each node v in the left-hand side of the inferred rule scheme, for each orbit type $\langle o \rangle$ we consider the point of interest $\texttt{center}(pos_{\langle o \rangle}(v))$. Reusing the notations introduced in Chapter 5, if d is a dart in a Gmap,

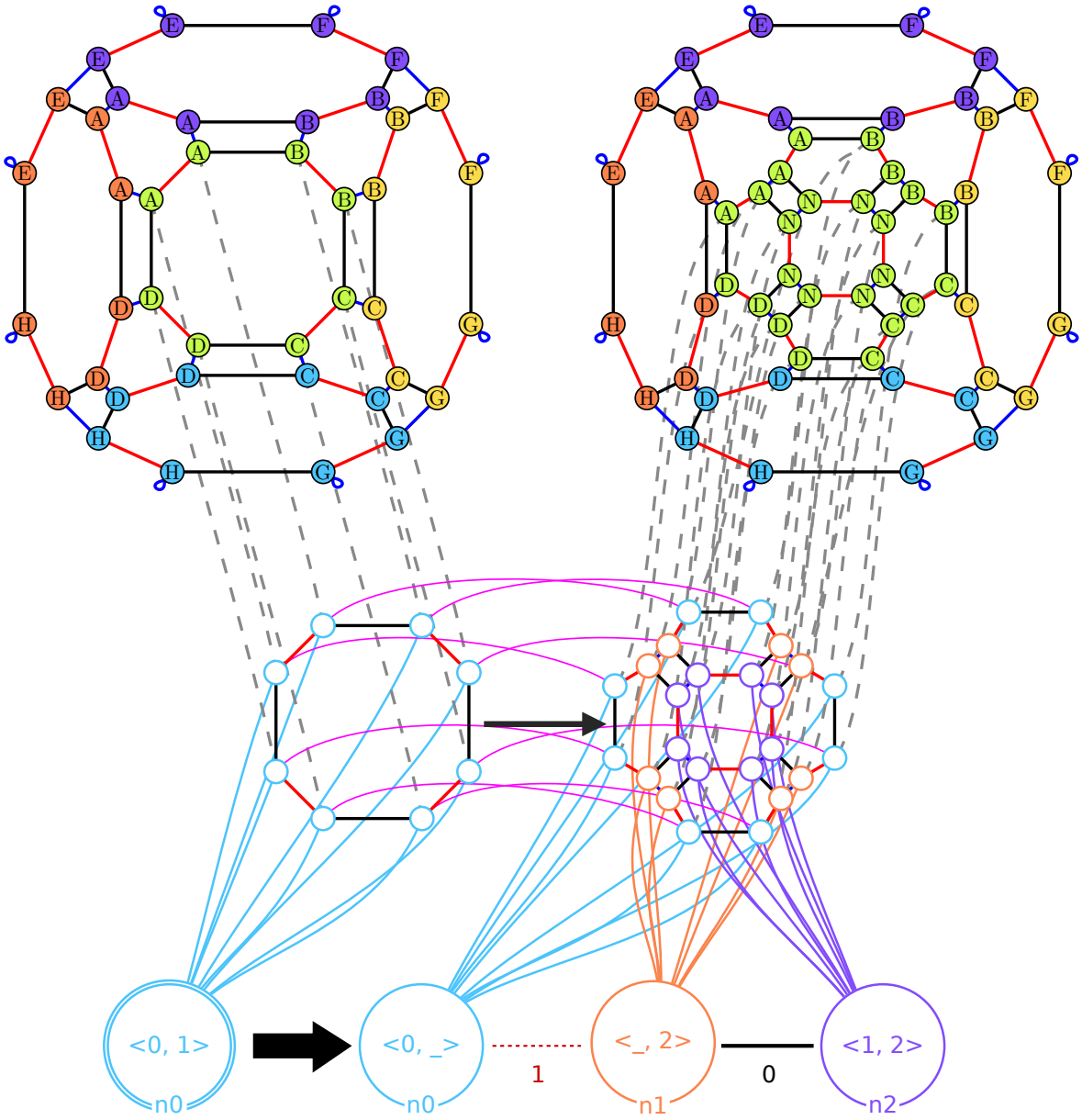


Figure 8.7: Context for the inference of embedding expressions on rule schemes.

`center`$(pos_{\langle o \rangle}(d))$ computes the barycenter of the positions of the vertices in the orbit $G\langle o \rangle(d)$. Here, d is replaced with n, where n is a left node of the rule scheme such that the computation can be instantiated regardless of the topology.

Orbit	Equivalent orbit
$\langle\rangle$	$\langle\rangle$, $\langle 1\rangle$, $\langle 2\rangle$, $\langle 3\rangle$, $\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 2,3\rangle$, $\langle 1,2,3\rangle$
$\langle 0\rangle$	$\langle 0\rangle$, $\langle 0,2\rangle$, $\langle 0,3\rangle$, $\langle 0,2,3\rangle$
$\langle 0,1\rangle$	$\langle 0,1\rangle$, $\langle 0,1,3\rangle$
$\langle 0,1,2\rangle$	$\langle 0,1,2\rangle$
$\langle 0,1,2,3\rangle$	$\langle 0,1,2,3\rangle$

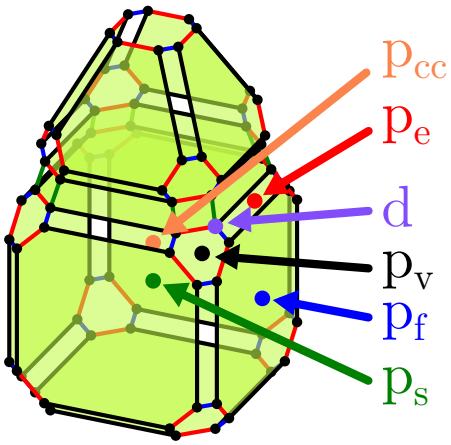
Table 8.1: Equivalent orbits for the position embedding.

If we consider a 3D object, 16 orbits are possible. However, as indicated in Table 8.1, only 5 orbits need to be considered, the other 11 being equivalent to one of them. Intuitively, the or-

bit $\langle 1,2,3\rangle$ induces an equivalence relation on the set of orbits (also depending on the cycle constraint from Definition 31), and we are choosing an orbit type per equivalence class. In other words, several orbit type yields the value when computing the barycenter of the collect for these orbits. Thus, we choose one orbit type for each distinct computation. We illustrate these points of interest in Figure 8.8.

Point of interest	Name	Embedding expression
vertex	p_v	$\texttt{center}(pos_{\langle\rangle}(d))$
edge	p_e	$\texttt{center}(pos_{\langle 0\rangle}(d))$
face	p_f	$\texttt{center}(pos_{\langle 0,1\rangle}(d))$
volume	p_s	$\texttt{center}(pos_{\langle 0,1,2\rangle}(d))$
cc	p_{cc}	$\texttt{center}(pos_{\langle 0,1,2,3\rangle}(d))$

(a)



(b)

Figure 8.8: Points of interest: (a) the list of points of interest and their expression, (b) computations of the points of interest on an object.

These points of interest can be computed for any dart. We write $p_v(d)$ for the vertex point of interest of a dart d, $p_e(d)$ for its edge point of interest, $p_f(d)$ for its face point of interest, $p_s(d)$ for its volume point of interest, and $p_{cc}(d)$ for its connected component point of interest. We generalize the notation as embedding expressions. Thus, we can use them as shortcuts on the nodes of a rule scheme. We write $\mathrm{Poi}(d)$ for the set of points of interest computed on dart d and extend it to $\mathrm{Poi}(v)$ for the set of embedding expressions based on the points of interest of node v.

Based on these points of interest, we can rewrite the equation 8.1 as follows:

$$pos(n_r) = \sum_{n_l \in \mathrm{V_L}} \sum_{p \in \mathrm{Poi}(n_l)} w_{p,n_l}\, p(n_l) + t \tag{8.2}$$

where $\mathrm{V_L}$ are the set of nodes of the rule scheme's left-hand side, (w_{p,n_l})'s are (unknown) weights and t encodes an (unknown) intrinsic translation.

The introduction of points of interest can also be seen as the addition of degrees of freedom to the mechanism. Indeed, directly adapting equation 8.1 to rule schemes would have yielded $pos(n_r) = \sum_{n_l \in \mathrm{V_L}} w_{n_l} p_v(n_l) + t$. Compared to equation 8.1, equation 8.2 is now purely symbolic since $pos(n_r)$ corresponds to the embedding expressions attached to node n_r. We will need to evaluate the equation on each dart associated with node n_r to obtain actual equations.

The general strategy for retrieving the embedding expression of the nodes from the right-hand side also has to be generalized.

1. For all nodes n_r in the right-hand, if n_r is a preserved node and all its associated darts have the same value in the after and before instances, then no computation needs to be realized, and we leave the expression of the right node empty. We also flag the node as solved.

2. For each solved node, we flag all nodes in it $\langle 1,2,3\rangle$-orbit as solved.

3. For all unsolved nodes, we build a system of equations based on its associated darts. Then, we try to solve the system. If the system admits a solution, we obtain an embedding expression for the node. Regardless of the existence of a solution, we flag it as solved and propagate the flag to its $\langle 1,2,3\rangle$-orbit.

We use a flag to avoid unnecessary computations since the application of a rule scheme propagates the computed embedding values. In step 3, if we cannot find a solution for a node, we will not find one for the nodes in its orbit since the darts necessarily have the same position (imposed by the embedding consistency, see Chapter 5). We reached a case that does not fit within our hypotheses. The core of this strategy is the equation system. The idea is to evaluate the equation 8.2 for each dart associated with the node. This evaluation mimics the instantiation of the rule scheme, replacing each point of interest with its value computed from the embedding values of the before instance. The weights (and the translation) remain unknown and constitute the variables of the equation system.

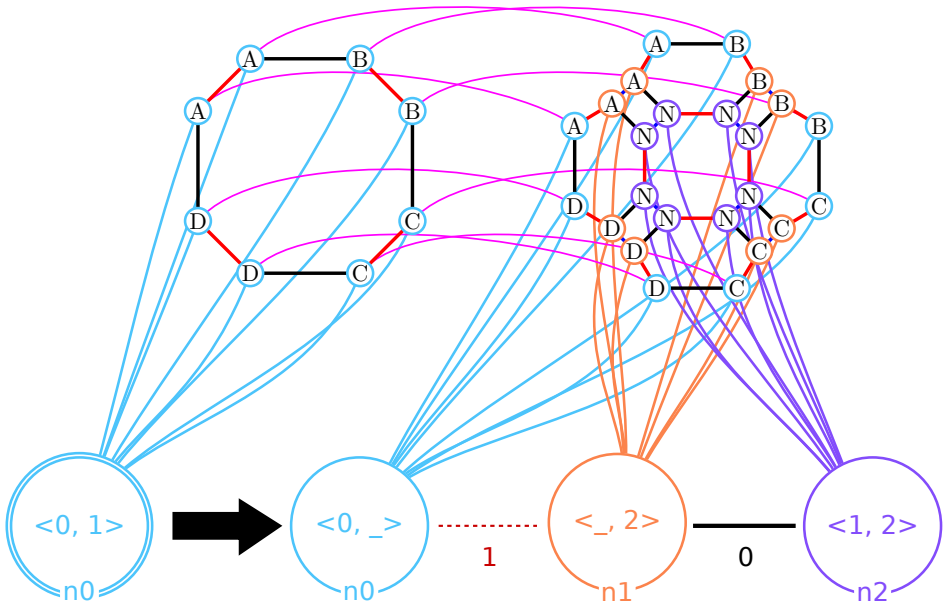


Figure 8.9: Resolution of the inference of embedding expressions.

We illustrate the construction and resolution of the inference of the position embedding with Figure 8.9. The operation preserves node $n0$. The darts associated with $n0$ have the same position in the before and after instances (the positions A, B, C, and D). Thus, we leave the expression on $n0$ empty and flag it as solved (step 1). The $\langle 1,2,3\rangle$-orbit of $n0$ contains $n1$ which is, therefore, also flagged as solved (step 2). This leaves node $n2$ to be processed by step 3. The left-hand side contains only one node, namely $n0$. Thus, the equation for $n2$ is:

$$pos(n2) = \underbrace{w_{(v,n0)}\,p_v(n0)}_{\text{vertex}} + \underbrace{w_{(e,n0)}\,p_e(n0)}_{\text{edge}} + \underbrace{w_{(f,n0)}\,p_f(n0)}_{\text{face}} + \underbrace{w_{(s,n0)}\,p_s(n0)}_{\text{volume}} + \underbrace{w_{(cc,n0)}\,p_{cc}(n0)}_{\text{connected component}} + t$$

We now evaluate this equation on all 8 darts associated with $n2$. For the example, we assume that A corresponds to the position $(0,0,0)$, B to $(1,0,0)$, C to $(1,1,0)$, D to $(0,1,0)$, and N

to $(0.5, 0.5, 0)$. Thus, the evaluation yields the system:

$$\begin{cases}
(0.5,0.5,0) = w_{(v,n0)} * (0,0,0) + w_{(e,n0)} * (0.5,0,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (1,0,0) + w_{(e,n0)} * (0.5,0,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (1,0,0) + w_{(e,n0)} * (1,0.5,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (1,1,0) + w_{(e,n0)} * (1,0.5,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (1,1,0) + w_{(e,n0)} * (0.5,1,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (0,1,0) + w_{(e,n0)} * (0.5,1,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (0,1,0) + w_{(e,n0)} * (0,0.5,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz) \\
(0.5,0.5,0) = w_{(v,n0)} * (0,0,0) + w_{(e,n0)} * (0,0.5,0) + w_{(f,n0)} * (0.5,0.5,0) + w_{(s,n0)} * (0.5,0.5,0) + w_{(cc,n0)} * (0.5,0.5,0) + (tx,ty,tz)
\end{cases}$$

Each equation is then projected on each coordinate x, y, z. In this case, we obtain a system of 24 equations with 8 variables, namely $w_{(v,n0)}$, $w_{(e,n0)}$, $w_{(f,n0)}$, $w_{(s,n0)}$, $w_{(cc,n0)}$, tx, ty, and tz. From the solver, we obtain the solution $w_{(v,n0)} = 0.0$, $w_{(e,n0)} = 0.0$, $w_{(f,n0)} = 1.0$, $w_{(s,n0)} = 0.0$, $w_{(cc,n0)} = 0.0$, $tx = 0.0$, $ty = 0.0$, and $tz = 0.0$.

Note that swapping $w_{(f,n0)} = 1.0$ for $w_{(f,n0)} = 0.0$ and choosing either one of $w_{(s,n0)} = 1.0$, $w_{(cc,n0)} = 1.0$, or $tx = 1.0$, $ty = 1.0$, and $tz = 1.0$ would have also yield a valid solution. We are exploiting the workflow of OR-tools that consider the variables iteratively. Thus, for this resolution, the solver never considered $w_{(s,n0)}$, $w_{(cc,n0)}$, tx, ty, or tz since $w_{(f,n0)} = 1.0$ provided a solution. From a practical perspective, we also use additional constraints to avoid degenerated solutions.

From the solution found by the CSP solver, we generate some code for the embedding expressions. This code exploits the script language of Jerboa [74], which intuitively reads like JAVA code. Here is the general pattern for the generated code of an inferred position expression:

```
Point3 res = new Point3( #translation# );                          // translation

// for all nodes in the inferred rule's left-hand side, written #node#
Point3 pV_#node# = Point3::middle(<>_position( #node# ));          // POI exportation
pV_#node#.scaleVect( #w(v,#node#)# );                              // computed weight
res.addVect(pV_#node#);

Point3 pE_#node# = Point3::middle(<0>_position( #node# ));         // POI exportation
pE_#node#.scaleVect( #w(e,#node#)# );                              // computed weight
res.addVect(pE_#node#);

Point3 pF_#node# = Point3::middle(<0,1>_position( #node# ));       // POI exportation
pF_#node#.scaleVect( #w(f,#node#)# );                              // computed weight
res.addVect(pF_#node#);

Point3 pS_#node# = Point3::middle(<0,1,2>_position( #node# ));     // POI exportation
pS_#node#.scaleVect( #w(s,#node#)# );                              // computed weight
res.addVect(pS_#node#);

Point3 pCC_#node# = Point3::middle(<0,1,2,3>_position( #node# ));  // POI exportation
pCC_#node#.scaleVect( #w(cc,#node#)# );                            // computed weight
res.addVect(pCC_#node#);

return res;
```

The generated expressions add the points of interest to the computed translation after scaling them based on the values obtained as solutions of the equation system. The expressions `Point3::middle(< #orb# >_position( #node# ))` compute the barycenter of the multiset

for the orbit type `#orb#` obtained from the node `#node#`. These expressions encode the exportation of the points of interest. Then, each point of interest is multiplied by the value of the solution obtained from the CSP solver, via the expressions `#poi#.scaleVect( #w(#poi#,#node#)# )`. In this expression, `#poi#` is the point of interest, `#node#` is the relevant node in the rule's left-hand side, and `#w(#poi#,#node#)#` is the weight obtained from the solution of the CSP. Finally, the computed contribution is added to the result via the expression `res.addVect(#poi#);`. Note that the result is initialized with the translation `#translation#` obtained in the solution of the CSP. Besides, if the weight obtained from the CPS equals zero for a point of interest (within a tolerance range), the corresponding part of the code is not generated.

As a last remark on the formalization of the problem as a constraint satisfaction problem, we also reduce the number of variables by exploiting the embedding consistency of orbits (see Chapter 5), similar to the flag mechanism. Indeed, any two nodes within the same $\langle 1,2,3\rangle$-orbit in the rule's left-hand side are associated with darts having the same positions. Thus, we can replace V_L in equation 8.2 by $V_L/\langle 1,2,3\rangle$, i.e., use only one node per $\langle 1,2,3\rangle$-orbit of the rule's left-hand side (see Definition 70 in Chapter 5).

8.3 Generalization to embeddings in a vector space

We explained the inference of embedding values for the position embedding attached to vertices. However, the exact same approach also works for other kinds of embedding. Indeed, the proposed solutions relied on affine combinations of values computed from points of interest. As long as the embedding values belong to a vector space, affine combinations are possible. Similarly, a vector space also allows for the computation of barycenters. Thus, points of interest can be extended to *values of interest*, which collect values based on an orbit type and computes the barycenter of the obtained multiset. From an implementation perspective, this extension is straightforward and boils down to two tasks. First, we encapsulate the embeddings as vector spaces encoding the sum of two vectors and the multiplication of a vector by a scalar. Secondly, we generalize the computation of the orbit types relevant to the values of interest based on the orbit type. Then, we can directly reuse the approach of Section 8.2.

We experimented with this approach on the color embedding, carried by the orbit type $\langle 0,1\rangle$, i.e., the faces of volumes. The orbit type $\langle 0,1\rangle$ for the embedding yields the orbit types $\langle\rangle$, $\langle 2\rangle$, $\langle 3\rangle$, $\langle 1,2\rangle$, $\langle 2,3\rangle$, $\langle 0,1,2\rangle$, $\langle 1,2,3\rangle$, and $\langle 0,1,2,3\rangle$ for the values of interest. The only issue raised is that RGB colors are represented by vectors in $[0,1]^3$ and not in $\mathbb{R}^3$. As a first solution, we propose to leave the inferred expression unchanged. In this case, rule applications might result in out-of-bond colors. This solution is suitable whenever the viewer has a solution to deal with out-of-bond values. As a second solution, we propose to automatically edit the inferred solution to clamp the result of the computation. Note that this modification occurs on the inferred expression and does not modify the resolution of the equation system. We can also return a default color or a random one as a final solution. This fallback also allows obtaining embedding expressions when the system cannot be solved.

This approach also allows inferring modifications of transparency, represented as a number in $[0,1]$. We will now present some results of our method, extending examples started in Chapter 7.

8.4 Results

In Chapter 7, we used the topological folding algorithm to infer the topological content of geometric modeling operations for terrain modeling and subdivision schemes. We explained that geometric values were manually added to the inferred rule schemes. We now discuss whether our inference mechanism retrieves suitable values for these expressions.

8.4.1 Applications for terrain modeling

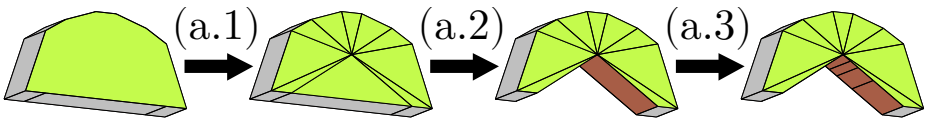


Figure 8.10: A procedural generation of natural arches.

We recall one of the procedural generations of arches inferred in Section 7.4.1 in Figure 8.10. All position computations are linear: barycenters of faces for step $a.1$, linear combinations of the position between the central edge and the base edges for step $a.3$. For the color, the subdivisions in steps $a.1$ and $a.3$ are obtained via color propagation and need no computation. In contrast, the brown color in step $a.2$ can be entirely encoded by the translation vector while all weights are put to zero. Thus, the complete inference of both the topology and the geometry can effectively be inferred.

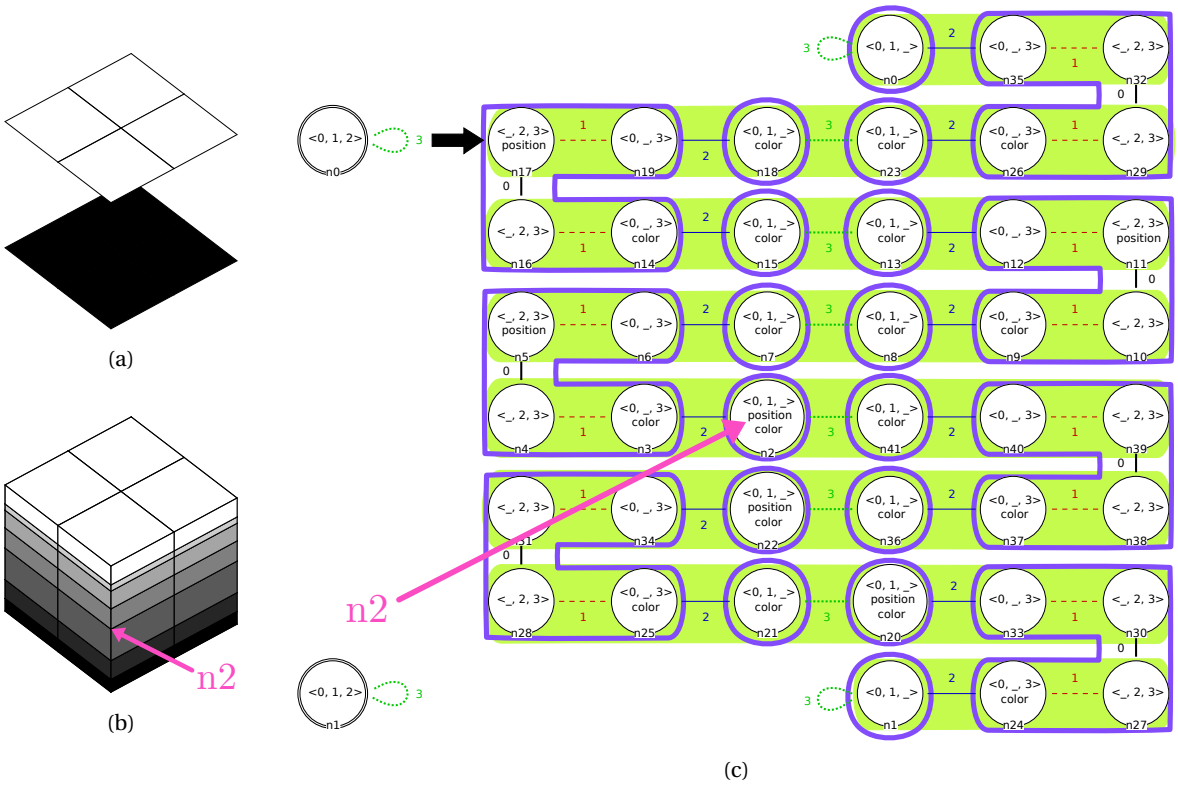


Figure 8.11: Inferring the position and color expressions for the layering operation: (a) before instance, (b) after instance, (c) inferred rule.

We experimented with the layering operation to include modifications of positions and colors. We replace the representative example used in Chapter 7 with that of Figure 8.11. The inferred rule

scheme is provided in Figure 8.11c. We highlighted in green the $\langle 1,2,3\rangle$-orbits and circled in purple the $\langle 0,1\rangle$-orbits. Exploiting the flags presented in Section 8.2.3, only one node needs a position expression in each green area. Similarly, only one node per purple area needs a color expression. The interlayers are interpolated from the initial surfaces. The new positions are computed as a weighted barycentric combination depending only on the vertex positions of interest obtained from the two nodes in the rule's left-hand side. Similarly, the colors are computed as weighted barycentric combinations of black and white (i.e., shades of grey). Node $n2$ (both in Figure 8.11b and 8.11c) describes a range of faces in an interlayer.

Solving the system for the position embedding yields the following expression:

```
// n2#position
Point3 res = new Point3(0.0,0.0,0.0);          // no translation
Point3 p0 = Point3::middle(<>_position(n1));   // position of vertex in the top surface
p0.scaleVect(0.5);                             // weight
res.addVect(p0);
Point3 p5 = Point3::middle(<>_position(n0));   // position of vertex in the bottom surface
p5.scaleVect(0.4999999999999999);              // weight
res.addVect(p5);
return res;
```

For the color embedding, we obtain the following expression.

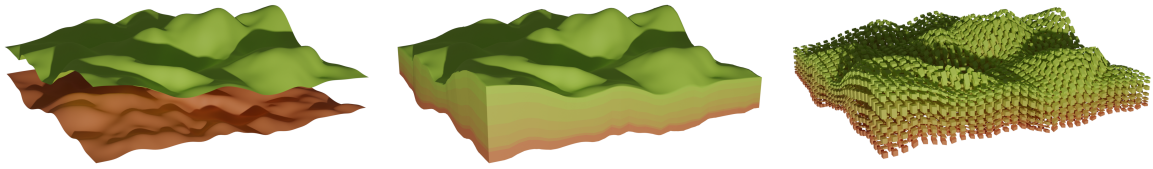
```
// n2#color
Color3 res = new Color3(0.0f,0.0f,0.0f);       // no (color) translation
Color3 c0 = Color3::middle(<>_color(n1));      // color of face in the top surface
c0.scaleVect(0.49803924560546875);             // weight
res.addVect(c0);
Color3 c8 = Color3::middle(<>_color(n0));      // color of face in the bottom surface
c8.scaleVect(0.5019607543945312);              // weight
res.addVect(c8);
res.clampVect();                               // clamp to avoid getting out of bound
return res;
```

Since we abstracted embedddings to be values in a vector space, all computations only rely on the functions `middle` that computes the barycenter of a multiset of values, `scaleVect` that multiplies the vector by a scalar, and `addVect` that sums two vectors. In other words, we obtain expressions similar to those explained in Section 8.2.3, replacing the points of interest with the suitable values of interest.

The complete inferred operation can then be applied to more complex surfaces, as displayed in Figure 8.12. The initial surfaces are given in Figure 8.12a, the result of the operation in Figure 8.12b, and the volume explosion of the result in Figure 8.12c where the initial surfaces have been hidden to reveal the inner volumes. A demo of this example is available online.[1]

For reference, the inference of the layering operation requires around 26ms for the topological folding algorithm and 549ms to infer the embedding expressions, i.e., solve all 25 systems of equations (6 for the positions and 19 for the colors).

[1] Link to the demo (last consulted on October 14th, 2022): `https://youtu.be/eI31_W1xF0I`.



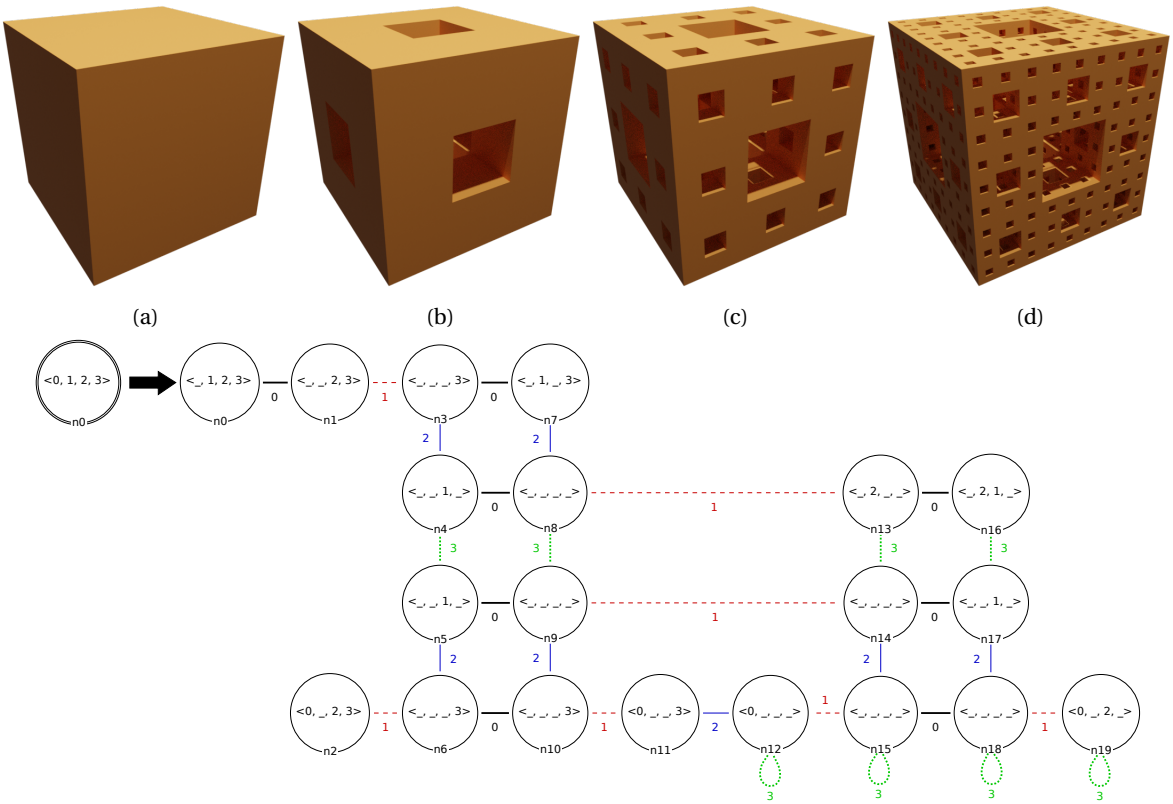
(a) (b) (c)

Figure 8.12: Application of the complete inferred layering operation to a complex scene: (a) initial surfaces, (b) result of the layering operation, and (c) volume explosion (after removal of the initial surfaces).

8.4.2 Application to subdivision schemes for volume refinement

We reconstructed two subdivision schemes for refining volumes. These operations work on 3D connected components, i.e., we used $\langle 0,1,2,3\rangle$ as inference parameter. For the color embedding, we used monochromatic objects, such that the inferred expressions are always trivial. Therefore, we will only discuss the inference of the position expressions.

Menger sponge



(a) (b) (c) (d)

(e)

Figure 8.13: Inferring the Menger sponge: a cube (a), the first (b), second (c) and third (d) iterations of the Menger sponge. The inferred operation (e) from the objects of Figures (a) and (b).

The Menger sponge is a 3D extension of the Cantor set (1D) or the Sierpinski carpet (2D). One refinement step can be obtained as follows. Take a cube and split each face into 9 squares to obtain 27 cubes. Remove all middle cubes (middle of faces and center of the initial cube). The 20 remaining cubes correspond to the iteration of the refinement step. This step is iterated on the newly obtained cubes. From the cube of Figure 8.13a, we obtain the first iteration of the Menger

sponge illustrated in Figure 8.13b. We can infer the operation by specifying that the operation occurs on the orbit $\langle 0,1,2,3\rangle$ to obtain the rule of Figure 8.13e. Note that an isolated cube is, for the purpose of inferring the Menger sponge, not an adequate topological description of a volume (see the discussion about inferring the layering operation on a one-face surface in Section 7.4.1). Thus, the topological folding algorithm requires at least two volumes to infer the operation. One solution is to infer with the first iteration as the before instance and the second iteration as the after instance. The solution we chose was to glue two cubes and perform the first iteration on the two cubes. Either approach solves the ambiguity on the role of dimension 3. Note that this inferred rule has 20 nodes on its right-hand side, which might already prove challenging to write (or read). We can now iterate the inferred operation and obtain the following iterations of the Menger sponge (second and third iterations in Figures 8.13c and 8.13d). We also inferred the operation directly producing the Menger sponge's second iteration. The rule has more than 400 nodes and is too large to be drawn properly.

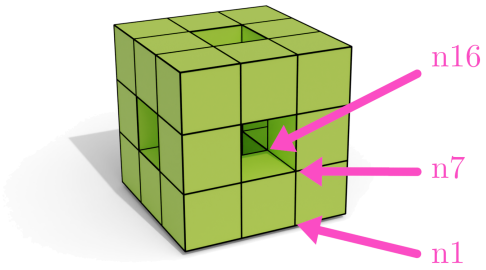


Figure 8.14: Vertices encoded by nodes $n1$, $n7$, and $n16$ from Figure 8.13e.

Since the orbit type $\langle 0,1,2,3\rangle$ is used as the rule scheme parameter, each node of the inferred rule scheme is associated with many darts from the before and after instance (48 to be precise). Therefore, only three position expressions need to be retrieved. These expressions are carried by the nodes $n1$, $n7$, and $n16$ in Figure 8.13e. Examples of such vertices associated with those nodes are given in Figure 8.14. The dart of reference for these three associated vertices belongs to the bottom-right vertex. We compare the inferred expressions with the ones used for generation.

Expression used for the generation:	Inferred expression:
Node $n1$	

`Point3 res = new Point3(0.0,0.0,0.0);`
`Point3 a = new Point3(n0.position);`
`a.scale(2.f/3.f);`
`res.add(a);`
`Point3 b = new Point3(n0@0.position);`
`b.scale(1.f/3.f);`
`res.add(b);`
`return res;`
	`Point3 res = new Point3(0.0,0.0,0.0);`
	`Point3 p0 = Point3::middle(<>_position(n0));`
	`p0.scale(0.3333333134651184);`
	`res.add(p0);`
	`Point3 p1 = Point3::middle(<0>_position(n0));`
	`p1.scale(0.6666666865348816);`
	`res.add(p1);`
	`return res;`

For this node, the two expressions are equivalent. Points a and $p0$ represent the same value computed differently. Point b corresponds to the other endpoint of the edge adjacent to $n0$, while $p1$ computes the midpoint of the edge. Therefore, $p1$ corresponds to $\frac{1}{2}(a+b)$. Thus, we obtain the

following equality.

$$\frac{2}{3}a+\frac{1}{3}b=\frac{1}{3}a+\frac{2}{3}(\frac{1}{2}a+\frac{1}{2}b)=\frac{1}{3}p0+\frac{2}{3}p1$$

Similarly, the expressions of nodes $n7$ and $n16$ were defined using neighbor accessors, but we infer expressions based on points of interest.

Expression used for the generation:	Inferred expression:
Node $n7$	

`Point3 res = new Point3(0.0,0.0,0.0);`
`Point3 a = new Point3(n0.position);`
`res.add(a);`
`Point3 b = new Point3(n0@0.position);`
`res.add(b);`
`Point3 c = new Point3(n0@1@0.position);`
`res.add(c);`
`res.scale(1.f/3.f);`
`return res;`
	`Point3 res = new Point3(0.0,0.0,0.0);`
	`Point3 p0 = Point3::middle(<>_position(n0));`
	`p0.scale(0.3333333134651184);`
	`res.add(p0);`
	`Point3 p2 =`
	`  Point3::middle(<0,1>_position(n0));`
	`p2.scale(0.6666666865348816);`
	`res.add(p2);`
	`return res;`

Node $n16$	

`Point3 res = new Point3(0.0,0.0,0.0);`
`Point3 b = new Point3(n0@0.position);`
`res.add(b);`
`Point3 c = new Point3(n0@1@0.position);`
`res.add(c);`
`Point3 d = new Point3(n0@2@1@0.position);`
`res.add(d);`
`res.scale(1.f/3.f);`
`return res;`
	`Point3 res = new Point3(0.0,0.0,0.0);`
	`Point3 p0 = Point3::middle(<>_position(n0));`
	`p0.scale(0.3333333134651184);`
	`res.add(p0);`
	`Point3 p3 =`
	`  Point3::middle(<0,1,2>_position(n0));`
	`p3.scale(0.6666666865348816);`
	`res.add(p3);`
	`return res;`

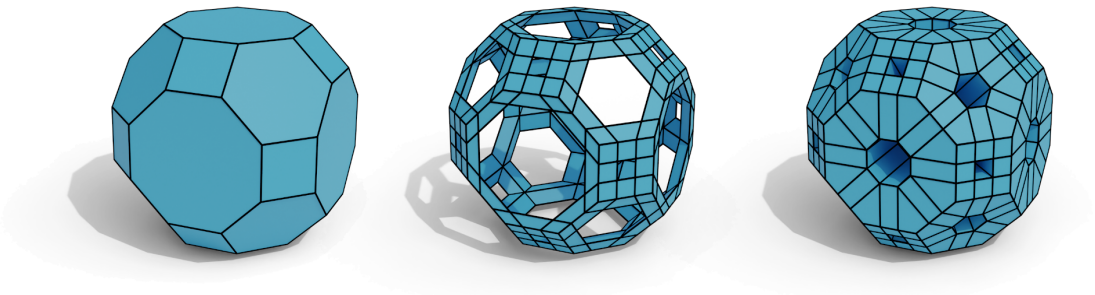


Figure 8.15: Two different geometric computations that yield the Menger sponge on a cube: (from left to right) the initial object, the object obtained with the rule used for the generation, and the result of the inferred operation.

Here again, the expressions are different, using path expressions for the already existing rule and points of interest for the inferred one. However, these expressions are not equivalent. Indeed, the expressions used for generation exploit the assumption that the modified object is a cube to compute the new positions from the endpoints of three edges incident to $n0$ (by rotating in the vertex). However, when modifying a solid with faces of higher arity, the two computations do not coincide. This difference is illustrated in Figure 8.15. Since the definition of the Menger sponge is always realized on the cube, both geometries are valid, and the final choice might be left to the aesthetic taste of the reader.

(2,2,2)-Menger sponge

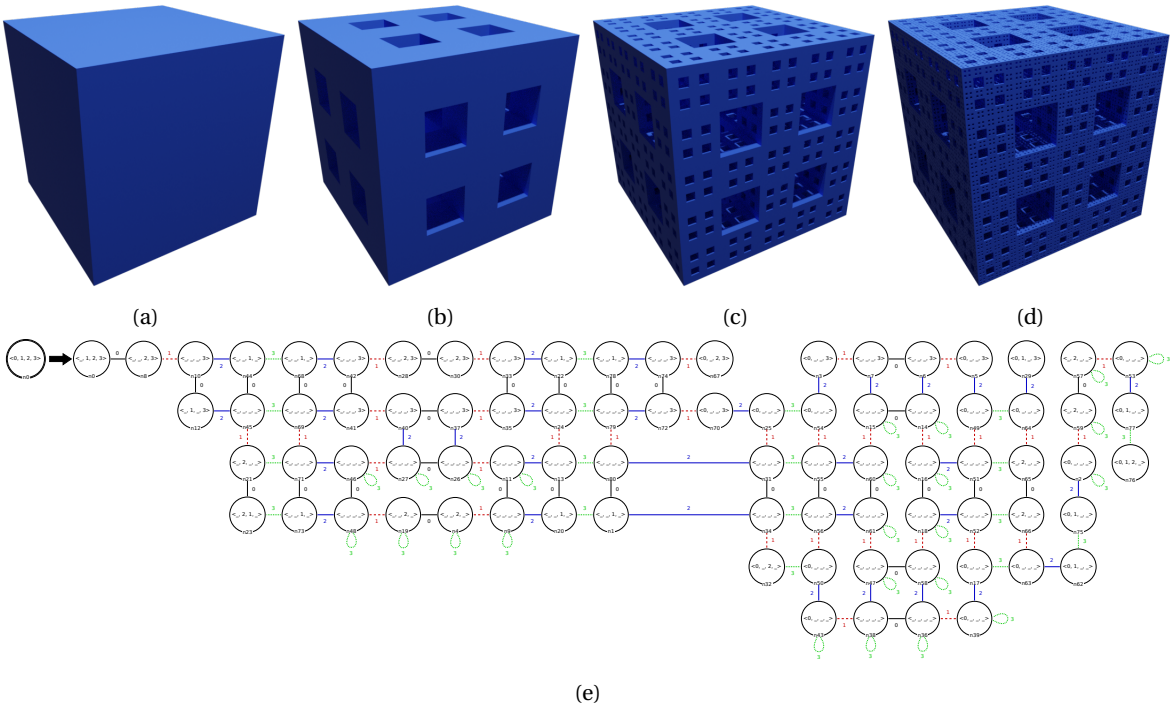


Figure 8.16: Inferring the (2,2,2)-Menger sponge: a cube (a), the first (b), second (c) and third(d) iterations of the (2,2,2)-Menger operation. The inferred operation (e).

In [154], the authors proposed a generalization of the Menger sponge to the Menger polycube. One iteration of the (L,M,N)-Menger operation transforms a polycube into a polycube with L holes along the x-axis, M holes along the y-axis, and N holes along the z-axis. Each hole has the same size as a one-unit cube and is separated from the nearest holes by a one-unit cube. From a cube, we built the first iteration of the (2,2,2)-Menger operation (see Figure 8.16b). The polycube is of genus 28 and consists of 81 volumes, 270 faces, 216 vertices. To our knowledge, there is no definition (either algorithmic or with a rule) of this operation. We manually built the polycube and used our algorithm to infer the operation. From the objects of Figures 8.16a and 8.16b, we inferred[2] the rule of Figure 8.16e. We used this operation to build the second and third iterations, respectively illustrated in Figures 8.16c and 8.16d. For information, it takes around 150ms to infer the topological part of the (2,2,2)-Menger operation and 700ms for the geometric expressions.

The complete set of inferred embedding expressions is given in Appendix D.1). The operation can be applied to any volume (see Figure 8.17).

[2] Here again, we used the trick of gluing two (poly)cubes to lift the ambiguity of the dimension 3.

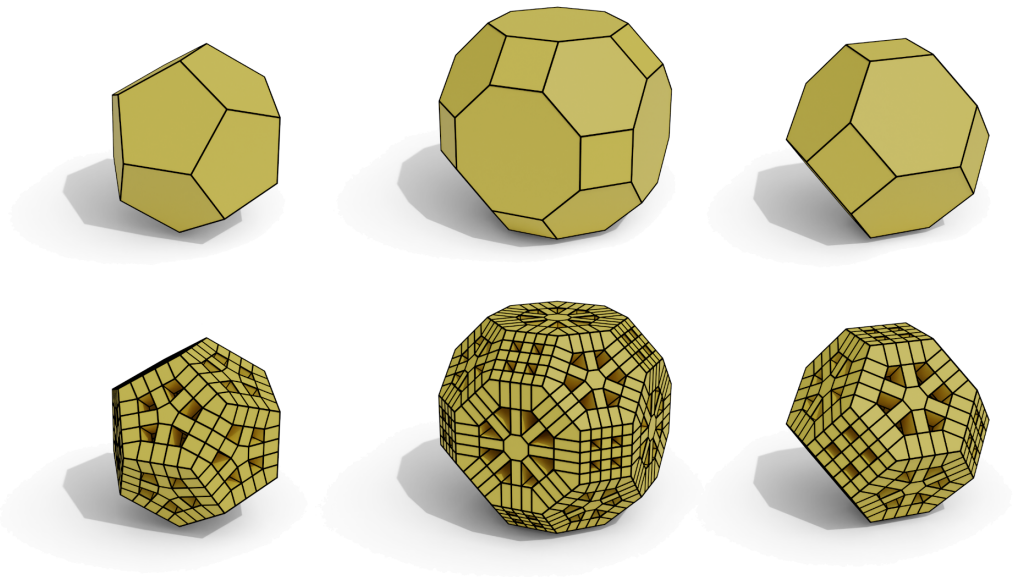


Figure 8.17: Application of the $(2,2,2)$-Menger operation to various solids.

8.5 Limits

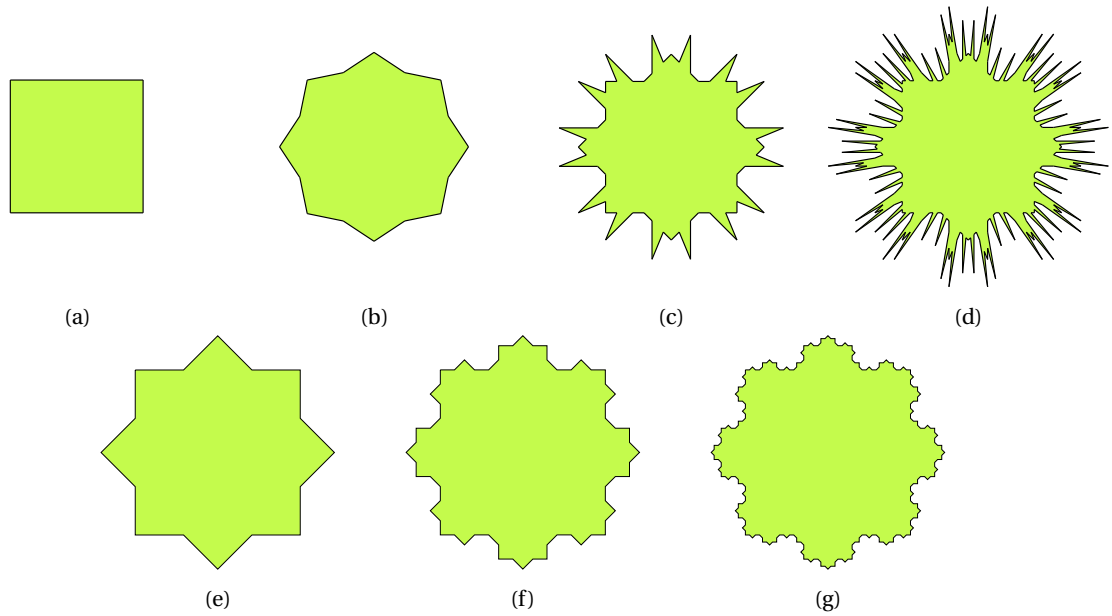


Figure 8.18: Inferring the von Koch's curve: (a) a square, (b) valid first iteration of the operation, (c) invalid second iteration of the operation obtained by applying the inferred operation, (d) invalid third iteration of the operation obtained by applying the inferred operation, (e) first iteration built with an L-system, (f) second iteration built with an L-system, (g) third iteration built with an L-system.

Sometimes, our hypotheses are too restrictive, and the system cannot be solved. However, it can also happen that the system admits a solution that does not correspond to the "valid" one. For instance, if we try to infer an operation similar to the von Koch's curve, we can solve the system and obtain a position expression. The inferred expression correctly reproduces the first iteration, but the subsequent iterations do not agree with the definition of the operation. The operation is illustrated in Figure 8.18. From the square in Figure 8.18a, we manually built the first iteration, displayed in Figure 8.18b. Two positions have to be computed when the operation is inferred on

the orbit type $\langle 0, 1 \rangle$. The first one belongs to the initial edges and can be properly computed. However, the second position is typically computed via a derivation angle from the original edge. Using points of interest, we compute this second position based on the vector between the edge midpoint and the face's barycenter. Thus, iterating the inferred operation translates the new vertices along these vectors. As illustrated in Figures 8.18c and 8.18d, we improperly compute some new positions and the errors accumulate. The desired iterations are displayed in Figures 8.18e to 8.18g. Interestingly, these iterations can be computed as L-systems, with the axiom $F+ +F+ +F+ +F$ and the derivation rule $F- > F-F+ +F-F$.

Summary of the chapter's contributions

In this chapter, we have seen a method to retrieve the embedding expressions on rule schemes, thus inferring the geometric modifications of geometric modeling operations. We assumed computations expressed as affine combinations such that transformations are essentially linear. The key idea was to abstract over the topology of the modified object via values of interest. In this chapter, we illustrated the notion with a barycentric approach, although other approaches exploiting neighbor accessors are conceivable.

Our work opens a new venue for the generation of modeling operations without programming knowledge. Indeed, inferring the geometric computations lead to a no-code development platform for topology-based geometric modeling. Compared to its topological counterpart, the inference of the geometry is highly non-deterministic as several computations may lead to the same values for a given input. In any case, the inference of the missing geometry will need to be generic to benefit from the orbit-based generalization.

Conclusion

Contributions

This thesis was realized at the intersection of two domains of computer science: graph rewriting and geometric modeling. More precisely, we used constructions and results from graph rewriting to describe a formalization of operations for geometric modeling. The constructed framework provided a guideline for the inference of operations.

Formalization of geometric modeling operations as graph transformation rules

The first part of this dissertation was dedicated to the foundations of our framework, where we presented a rule-based language for the design of topology-based geometric modeling operations. In Chapter 3, we extended the graph representation of Gmaps to obtain a unified model also encoding Omaps. This extension from Gmap to Omap resulted from refining the topological constraints defining Gmaps. We superimposed local variations of the incident arcs, non-orientation, and cycle constraints on graphs arc-labeled with dimensions to obtain both models. From this representation of subdivided objects as graphs, we formalized modeling operations with DPO rules. Local necessary and sufficient conditions on rules were presented, such that checking some properties on the neighboring elements of a node ensured the preservation of a constraint. The obtained rules suffered from an overspecialization in the sense that semantically equivalent operations were described by distinct rules based on the underlying topology of the object. In other words, there was a rule for each possible application of a geometric modeling operation rather than a rule effectively describing the operation. In a sense, geometric modeling as an application domain of graph transformations brings the need for more abstract and general rules, which are not usual within the community of graph rewriting. Therefore, we extended DPO rules in Chapter 4 via a functorial approach exploiting a product of graphs simulating relabeling functions and describing global operations. From this extension, we obtained rule schemes parameterized by a set of words describing paths in the modified object. Instead of matching the rule scheme directly to the topological graph, we first extracted a pattern graph via identified nodes called hooks. We also lifted the topological conditions from rules to rule schemes and gave algorithms to check these conditions via traversals of the rule scheme. In Chapter 5, we extended this topological description with geometric information handled as attributes on nodes of the graphs to describe embedding functions usually conceived as acting on topological cells. The representation of functions on subgraphs was obtained via constraints on the attributed graphs, added to the already existing topological one. We derived conditions on rules to ensure the preservation of the geometric constraints on the graphs. Since those conditions imposed that complete orbits should be matched to modify

the embedding value, we described a rule completion mechanism via a topological extension and an embedding propagation. This completion mechanism allowed for a more compact representation of geometric modifications while restoring geometric consistency on rules.

Inference of geometric modeling operations with Jerboa

In the second part of the thesis, we focused on the inference of geometric modeling operations from a representative example. This inference mechanism essentially answered two needs. Prosaically, it enabled the design of modeling operations in the Jerboa platform without requiring in-depth knowledge of its domain-specific language. With a more general ambition, we provided the first solution to retrieve operations with topological guarantees on their application. Although Jerboa's rule-based language can now effectively be hidden from the user, it provided some prior structure to comply with when trying to retrieve the operations. More precisely, our approach exploits the regularity of Gmaps and the genericity of rule schemes to deduce operations parameterized by an orbit type. The topological folding algorithm presented in Chapter 7 performs a traversal of a partial mapping between the nodes of two Gmaps trying to find isomorphic orbits up to relabeling. The algorithm can be seen as the reverse process of applying a rule or as a graph quotient.

We extended this topological inference with a mechanism to retrieve embedding expressions directly on rule schemes, exploiting information from the topological folding algorithm. We tackled the problem by formulating a family of functions and values of interest. These notions are considered a generic approach to the problem that can be embodied differently based on the known hypotheses on the operation to infer. We illustrated these notions with affine combinations for the family of functions and orbit barycenters for the values of interest, retrieving embedding expressions for the positions and colors. We tested our method with examples from terrain modeling and subdivision schemes. In particular, we obtained a description of the $(2,2,2)$-Menger operation [154], which (to our knowledge) did not previously admit any algorithmic description (or any implementation).

We presented theoretical and practical guarantees about the algorithm and implemented it in Jerboa, obtaining a new tool called Jerboa Studio.

Perspectives

About graph transformations for geometric modeling

The content of this dissertation and our representation of geometric modeling operations raise some interesting questions related to the ambition of creating a unified framework for topology-based geometric modeling. Such a framework would require some technical harmony while staying user-friendly. In particular, a domain-specific language can inherently represent a degree of technicality which might be challenging to fit with a needed simplicity of manipulation.

- How can we formalize geometric modifications on Omaps? The first part of Chapter 5 can directly be adapted to Omaps with a slight adjustment on the condition related to orbits, i.e., by replacing arcs with paths, essentially as we did for pattern graphs. However, the completion mechanism would have to be modified since the completion of the topological cells

would result in disconnected graphs, hindering the possibility of propagating the embeddings.

- Can we generalize the framework to rule schemes? The second question has already partly been answered by Thomas Bellet in his Ph.D. thesis [13], restricting the study to rule schemes on Gmaps with topological variables where all embedding expressions exploit a construction similar to our identifier terms. However, the exact consistency conditions remain an open question. In particular, instantiated rules may have orbits embedded with different yet equivalent terms. One possibility could be to consider equivalence classes on terms to use as node attributes in the rules.

About the inference of geometric modeling operations

The methods presented in this document enable the complete inference of geometric modeling operations under the hypothesis of linearity for the embedding modifications. The overall process can still be improved. For instance, we might want to improve the topological part of the inference or, more precisely, extend the preprocessing of the algorithm.

- Using Jerboa's features, we can simplify the task of building the mapping used as input of the inference mechanism. For instance, we can deduce it from the node identifiers if snapshots of the graphs are given, or we can reconstruct it based on the mapping of orbits. Still, the automated reconstruction of the complete mapping would be a real asset to help users design operations.

- Our inference mechanism might also benefit from more robustness to noise on the representative example. Indeed, the topological folding algorithm essentially tries to find local redundancies that can be abstracted over to obtain general operations. However, if the object presents a local anomaly, such as a default of symmetry on the before instance, or if the transformation has not been entirely realized on the after instance, then our algorithm will fail to generate a general rule scheme. Although we will always find a solution with the empty orbit type, it would be a real asset if we could discard such irregularities before running the algorithm. We could even run the algorithm as a repair mechanism while inferring the operation, i.e., when the algorithm enters a failure state, we let it run, correcting the input to try to fit the scheme being inferred. This question essentially deals with the issue of inferring geometric modeling operations in the context of incomplete information.

Regarding the geometric part of our inference mechanism, we believe our approach would benefit from further development and enrichment of our solution.

- We used affine combination as an extension of purely barycentric computations such that operations that assign a value could be retrieved while also enabling global translations in the operations. We could transform this inherent transformation into a global variable on the rule. This approach raises questions regarding the pertinence of this approach. In particular, we would want to reduce the number of global variables by finding redundant ones through the nodes. In a sense, we would transform the systems of equations solved separately into a global system where all equations are solved simultaneously.

- More generally, we might want to get rid of the linearity hypothesis, thus changing the family of functions, e.g., to polynomials of bounded degree.
- The subdivision schemes for surfaces studied in Chapter 3 exploited geometric expressions that are typically retrieved via the neighbor accessors (the functions `@i` for dimensions i in $0..n$). Modifying the values of interest to exploit these accessors would allow retrieving most subdivision schemes. This question relates to a more general one regarding which computations would be more suited for inferring geometric expressions. However, there might probably not be an ultimate best solution but only local best ones, based on the task at hand.
- We might also want to study correlations between embeddings, e.g., barycenters with weights computed from colors or translations based on normal vectors. Such approaches would allow solving the inference of the von Koch operation (see Section 8.5). This concern also raises questions about extending our approach to embedding values not represented with vector spaces, such as orientation booleans.

About helping the design of geometric modeling operations

On a somewhat different subject, the goal of Jerboa Studio is to help design geometric modeling operations within the domain-specific language of Jerboa. However, Jerboa's language may appear difficult to read even when already written. Besides, conceiving an operation can be difficult because the user might not be confident of the desired result when the operation is applied to a different object. They might be able to produce one (or a few) representative examples of the operation but would like to visualize it on other objects. A solution to display an object on which the inferred rule can effectively be applied (and the result of the rule application) might help a user understand the result of the inference. Such a solution requires generating a Gmap based on constraints described by the rule scheme's left-hand side. While an indisputable asset for Jerboa, this topic also presents an interesting research question, entangling the question of designing a dedicated graph grammar based on constraints for the topological part with more practical questions raised from the reconstruction of embedding values associated with such generated objects.

Taking into account an automated solution for the before-to-after-instance mapping, we would obtain a tool where all theoretical parts supporting both the representation of the geometric objects and the modeling operations could be hidden. The domain-expert user would only see and manipulate objects. Borrowing ideas from sketch modeling [134], one could even imagine a tool where all modeling operations could be obtained and applied via simple drawings.

Beyond the inference of geometric modeling operations

In this work, we presented a method to infer geometric modeling operations. We made several choices and hypotheses along the way. We exploited

- the combinatorial model of Gmaps that separates the topology and the geometry in the representation of the objects,
- the double-pushout approach to graph transformations for the representations of geometric modeling operations,

- a pattern-based generalization of modeling operations,
- Jerboa's rule language as a practical formalism for these generalized rules,
- linearity on the embedding expressions,

These choices and hypotheses constitute foundations supporting our inference framework. Changing the paradigm opens the venue for other inference methods both within the field of geometric modeling and graph rewriting.

Bibliography

[1] Abdullah Alshanqiti, Reiko Heckel, and Timo Kehrer. "Visual contract extractor: a tool for reverse engineering visual contracts using dynamic analysis". In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. New York, NY, USA: Association for Computing Machinery, Aug. 25, 2016, pp. 816–821. ISBN: 978-1-4503-3845-5. DOI: 10.1145/2970276.2970287 (cit. on p. 3).

[2] Jakob L. Andersen, Christoph Flamm, Daniel Merkle, and Peter F. Stadler. "A Software Package for Chemically Inspired Graph Transformation". In: *Graph Transformation*. Ed. by Rachid Echahed and Mark Minas. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 73–88. ISBN: 978-3-319-40530-8. DOI: 10.1007/978-3-319-40530-8_5 (cit. on pp. 77, 186).

[3] Agnès Arnould, Hakim Belhaouari, Thomas Bellet, Pascale Le Gall, and Romain Pascual. "Preserving consistency in geometric modeling with graph transformations". In: *Mathematical Structures in Computer Science* (Oct. 18, 2022). Publisher: Cambridge University Press, pp. 1–48. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129522000226. (Visited on 10/19/2022) (cit. on pp. 8, 124, 126, 129, 138, 139, 147, 158, 168, 174, 200, 297).

[4] László Babai. "Automorphism groups, isomorphism, reconstruction". In: *Handbook of combinatorics (vol. 2)*. Cambridge, MA, USA: MIT Press, Mar. 1, 1996, pp. 1447–1540. ISBN: 978-0-262-07171-0 (cit. on p. 186).

[5] László Babai and Eugene M. Luks. "Canonical labeling of graphs". In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. STOC '83. New York, NY, USA: Association for Computing Machinery, Dec. 1, 1983, pp. 171–183. ISBN: 978-0-89791-099-6. DOI: 10.1145/800061.808746 (cit. on p. 186).

[6] Ilya Baran and Jovan Popović. "Automatic rigging and animation of 3D characters". In: *ACM Transactions on Graphics* 26.3 (July 29, 2007), 72–es. ISSN: 0730-0301. DOI: 10.1145/1276377.1276467 (cit. on p. 12).

[7] Michael Barr and Charles Wells. *Category theory for computing science*. Vol. 49. New York: Prentice Hall, 1990 (cit. on pp. 23, 24, 31).

[8] Michel Bauderon. "Parallel Rewriting of Graphs through the Pullback Approach". In: *Electronic Notes in Theoretical Computer Science*. SEGRAGRA 1995 2 (Jan. 1, 1995), pp. 19–26. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80176-8 (cit. on p. 87).

[9] Bruce G. Baumgart. *Winged edge polyhedron representation*. Stanford, Calif.: Stanford University, Oct. 1972 (cit. on p. 14).

[10] Bruce G. Baumgart. “A polyhedron representation for computer vision”. In: *Proceedings of the May 19-22, 1975, national computer conference and exposition*. AFIPS ’75. New York, NY, USA: Association for Computing Machinery, May 19, 1975, pp. 589–596. ISBN: 978-1-4503-7919-9. DOI: 10.1145/1499949.1500071 (cit. on p. 14).

[11] Nicolas Behr and Pawel Sobociński. “Rule Algebras for Adhesive Categories”. In: *Logical Methods in Computer Science* Volume 16, Issue 3 (July 3, 2020). DOI: 10.23638/LMCS-16(3:2)2020. arXiv: 1807.00785 (cit. on pp. 34, 37).

[12] Hakim Belhaouari, Agnès Arnould, Pascale Le Gall, and Thomas Bellet. “Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling”. In: *Graph Transformation*. ICGT 2014. Ed. by Holger Giese and Barbara König. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 269–284. ISBN: 978-3-319-09108-2. DOI: 10.1007/978-3-319-09108-2_18 (cit. on pp. 2, 43, 56, 58, 81, 84, 85, 117, 179, 189, 199).

[13] Thomas Bellet. “Transformations de graphes pour la modélisation géométrique à base topologique”. These de doctorat. Poitiers, July 10, 2012. URL: https://www.theses.fr/2012POIT2261 (visited on 08/19/2020) (cit. on p. 268).

[14] Thomas Bellet, Agnès Arnould, Hakim Belhaouari, and Pascale Le Gall. “Geometric Modeling: Consistency Preservation Using Two-Layered Variable Substitutions”. In: *Graph Transformation (ICGT 2017)*. Ed. by Juan de Lara and Detlef Plump. Vol. 10373. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 36–53. ISBN: 978-3-319-61470-0. DOI: 10.1007/978-3-319-61470-0_3 (cit. on pp. 43, 77, 84, 129, 139, 179, 188, 189, 198).

[15] Thomas Bellet, Agnès Arnould, and Pascale Le Gall. “Rule-based transformations for geometric modelling”. In: *6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), Part of ETAPS 2011*. Vol. 48. Saarbrücken, Germany, Apr. 2011, pp. 20–37. DOI: 10.4204/EPTCS.48.5 (cit. on pp. 43, 77, 129, 139).

[16] Thomas Bellet, Mathieu Poudret, Agnès Arnould, Laurent Fuchs, and Pascale Le Gall. “Designing a Topological Modeler Kernel: A Rule-Based Approach”. In: *2010 Shape Modeling International Conference*. 2010 Shape Modeling International Conference. June 2010, pp. 100–112. DOI: 10.1109/SMI.2010.31 (cit. on pp. 43, 47, 56, 58, 81, 84, 125, 181, 195, 199).

[17] Fatma Ben Salah, Hakim Belhaouari, Agnès Arnould, and Philippe Meseure. “A general physical-topological framework using rule-based language for physical simulation”. In: *12th International Conference on Computer Graphics Theory and Application (VISIGRAPP/GRAPP 2017)*. Vol. GRAPP. VISIGRAPP 2017 proceedings. Porto, Portugal, Feb. 2017. DOI: 10.5220/0006119802200227 (cit. on pp. 4, 17, 201).

[18] Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Gael Guennebaud, Joshua Levine, Andrei Sharf, and Claudio Silva. “A Survey of Surface Reconstruction from Point Clouds”. In: *Computer Graphics Forum* (2016), p. 27. DOI: 10.1111/cgf.12802 (cit. on p. 12).

[19] Enrico Biermann, Claudia Ermel, and Gabriele Taentzer. “Formal foundation of consistent EMF model transformations by algebraic graph transformation”. In: *Software & Systems Modeling* 11.2 (May 1, 2012), pp. 227–250. ISSN: 1619-1374. DOI: 10.1007/s10270-011-0199-7 (cit. on p. 77).

[20] Paul Boehm, Harald-Reto Fonio, and Annegret Habel. “Amalgamation of graph transformations: A synchronization mechanism”. In: *Journal of Computer and System Sciences* 34.2 (Apr. 1, 1987), pp. 377–408. ISSN: 0022-0000. DOI: 10.1016/0022-0000(87)90030-4 (cit. on p. 84).

[21] Evans Bohl, Olivier Terraz, and Djamchid Ghazanfarpour. “Modeling fruits and their internal structure using parametric 3Gmap L-systems”. In: *The Visual Computer* 31.6 (June 1, 2015), pp. 819–829. ISSN: 1432-2315. DOI: 10.1007/s00371-015-1108-9 (cit. on pp. 2, 200).

[22] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. “Quad-Mesh Generation and Processing: A Survey”. In: *Computer Graphics Forum* 32.6 (2013), pp. 51–76. ISSN: 1467-8659. DOI: 10.1111/cgf.12014 (cit. on p. 82).

[23] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Ed. by Ronald V. Book and Friedrich Otto. Text and Monographs in Computer Science. New York, NY: Springer, 1993. ISBN: 978-1-4613-9771-7 (cit. on pp. 106, 109, 110, 116).

[24] Francis Borceux. *Handbook of categorical algebra: volume 1, Basic category theory*. Vol. 1. Cambridge University Press, 1994 (cit. on p. 24).

[25] Mario Botsch, Stefan Steinberg, Stefan Bischoff, and Leif Kobbelt. “OpenMesh: A Generic and Efficient Polygon Mesh Data Structure”. In: *OpenSG Symposium 2002*. 2002 (cit. on p. 14).

[26] Pierre Bourquat, Hakim Belhaouari, Philippe Meseure, Valentin Gauthier, and Agnès Arnould. “Transparent Parallelization of Enrichment Operations in Geometric Modeling:” in: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. 15th International Conference on Computer Graphics Theory and Applications. Valletta, Malta, 2020, pp. 125–136. ISBN: 978-989-758-402-2. DOI: 10.5220/0008965701250136 (cit. on pp. 121, 200).

[27] Kristopher Brown, Evan Patterson, Tyler Hanks, and James Fairbanks. “Computational Category-Theoretic Rewriting”. In: *Graph Transformation*. Ed. by Nicolas Behr and Daniel Strüber. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 155–172. ISBN: 978-3-031-09843-7. DOI: 10.1007/978-3-031-09843-7_9 (cit. on p. 43).

[28] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. “Designing and proving correct a convex hull algorithm with hypermaps in Coq”. In: *Computational Geometry: Theory and Applications*. Geometric Constraints and Reasoning 45.8 (Oct. 1, 2012), pp. 436–457. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2010.06.006 (cit. on p. 17).

[29] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. “Directed Edges—A Scalable Representation for Triangle Meshes”. In: *Journal of Graphics Tools* 3.4 (Jan. 1, 1998), pp. 1–11. ISSN: 1086-7651. DOI: 10.1080/10867651.1998.10487494 (cit. on p. 14).

[30] Anaïs Cardot, David Marcheix, Xavier Skapin, Agnès Arnould, and Hakim Belhaouari. "Persistent Naming Based on Graph Transformation Rules to Reevaluate Parametric Specification". In: *Computer-Aided Design and Applications* 16.5 (Jan. 21, 2019), pp. 985–1002. ISSN: 16864360. DOI: 10.14733/cadaps.2019.985-1002 (cit. on pp. 2, 200, 208).

[31] E. Catmull and J. Clark. "Recursively generated B-spline surfaces on arbitrary topological meshes". In: *Computer-Aided Design* 10.6 (Nov. 1, 1978), pp. 350–355. ISSN: 0010-4485. DOI: 10.1016/0010-4485(78)90110-0 (cit. on pp. 148, 236).

[32] Chaomei Chen. *Information Visualization: Beyond the Horizon*. 2nd ed. Springer London, Oct. 28, 2004. 386 pp. ISBN: 978-1-85233-789-6. DOI: 10.1007/1-84628-579-8 (cit. on p. 223).

[33] Chris Conlan. *The Blender Python API: Precision 3D Modeling and Add-on Development*. 1st ed. Berkeley, CA: Apress, June 14, 2017. XX, 138. ISBN: 978-1-4842-2802-9. DOI: 10.1007/978-1-4842-2802-9 (cit. on p. 207).

[34] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. "A (sub)graph isomorphism algorithm for matching large graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.10 (Oct. 2004), pp. 1367–1372. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2004.75 (cit. on p. 186).

[35] A. Corradini, U. Montanari, and F. Rossi. "Graph processes". In: *Fundamenta Informaticae* 26.3 (June 1, 1996), pp. 241–265. ISSN: 0169-2968. DOI: 10.3233/FI-1996-263402 (cit. on p. 35).

[36] Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. "Sesqui-Pushout Rewriting". In: *Graph Transformations*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 30–45. ISBN: 978-3-540-38872-2. DOI: 10.1007/11841883_4 (cit. on p. 210).

[37] Bruno Courcelle. "The expression of graph properties and graph transformations in monadic second-order logic". In: *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. USA: WORLD SCIENTIFIC, Feb. 1997, pp. 313–400. ISBN: 978-981-02-2884-2. DOI: 10.1142/9789812384720_0005 (cit. on p. 51).

[38] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 2012. ISBN: 978-0-521-89833-1. DOI: 10.1017/CBO9780511977619 (cit. on p. 51).

[39] Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. "Digital geometry processing with discrete exterior calculus". In: *ACM SIGGRAPH 2013 courses*. SIGGRAPH '13. New York, NY, USA: ACM, 2013, pp. 1–126 (cit. on p. 13).

[40] Benoit Crespin, Richard Bézin, Xavier Skapin, Olivier Terraz, and Philippe Meseure. "Generalized maps for erosion and sedimentation simulation". In: *Computers & Graphics* 45 (2014), pp. 1–16. ISSN: 0097-8493. DOI: 10.1016/j.cag.2014.07.001 (cit. on p. 234).

[41] G. Damiand, Christine Solnon, C. D. L. Higuera, J. Janodet, and Émilie Samuel. "Polynomial algorithms for subisomorphism of nD open combinatorial maps". In: *Comput. Vis. Image Underst.* (2011). DOI: 10.1016/j.cviu.2010.12.013 (cit. on p. 187).

[42] Guillaume Damiand. "Combinatorial Maps". In: *CGAL User and Reference Manual*. 5.5. CGAL Editorial Board, 2022. URL: https://doc.cgal.org/5.5/Manual/packages.html#PkgCombinatorialMaps (cit. on p. 6).

[43] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, Sept. 19, 2014. 407 pp. ISBN: 978-1-4822-0652-4 (cit. on pp. 4, 5, 14, 15, 50, 54, 182).

[44] Guillaume Damiand and Vincent Nivoliers. "Query-replace operations for topologically controlled 3D mesh editing". In: *Computers & Graphics* (June 18, 2022). ISSN: 0097-8493. DOI: 10.1016/j.cag.2022.06.008 (cit. on pp. 18, 187).

[45] Christophe Dehlinger and Jean-François Dufourd. "Formal specification and proofs for the topology and classification of combinatorial surfaces". In: *Computational Geometry* 47.9 (Oct. 1, 2014), pp. 869–890. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2014.04.007 (cit. on p. 17).

[46] Edsger Wybe Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976 (cit. on p. 75).

[47] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. "Hoppity: Learning Graph Transformations to Detect and Fix Bugs in Programs". In: International Conference on Learning Representations (ICLR). 2020, p. 17 (cit. on p. 3).

[48] Daniel Doo and Malcolm A. Sabin. "Behaviour of recursive division surfaces near extraordinary points". In: *Computer-Aided Design* 10.6 (Nov. 1, 1978), pp. 356–360. ISSN: 0010-4485. DOI: 10.1016/0010-4485(78)90111-2 (cit. on p. 239).

[49] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. "InverseCSG: automatic conversion of 3D models to CSG trees". In: *ACM Transactions on Graphics* 37.6 (Dec. 4, 2018), 213:1–213:16. ISSN: 0730-0301. DOI: 10.1145/3272127.3275006 (cit. on p. 209).

[50] Johannes Dyck and Holger Giese. "Inductive Invariant Checking with Partial Negative Application Conditions". In: *Graph Transformation (ICGT 2015)*. Ed. by Francesco Parisi-Presicce and Bernhard Westfechtel. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 237–253. DOI: 10.1007/978-3-319-21145-9_15 (cit. on p. 76).

[51] Nira Dyn, David Levine, and John A. Gregory. "A butterfly subdivision scheme for surface interpolation with tension control". In: *ACM Transactions on Graphics* 9.2 (Apr. 1990), pp. 160–169. ISSN: 0730-0301. DOI: 10.1145/78956.78958 (cit. on pp. 17, 167, 237).

[52] Jürgen Ebert and Tassilo Horn. "GReTL: an extensible, operational, graph-based transformation language". In: *Software & Systems Modeling* 13.1 (Feb. 2014), pp. 301–321. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-012-0250-3 (cit. on p. 42).

[53] John Robert Edmonds. "A combinatorial representation for oriented polyhedral surfaces". PhD thesis. University of Maryland, 1960 (cit. on p. 13).

[54] H. Ehrig and A. Habel. “Graph Grammars with Application Conditions”. In: *The Book of L*. Ed. by G. Rozenberg and A. Salomaa. Berlin, Heidelberg: Springer, 1986, pp. 87–100. ISBN: 978-3-642-95486-3. DOI: 10.1007/978-3-642-95486-3_7 (cit. on p. 71).

[55] Hartmut Ehrig. “Introduction to the algebraic theory of graph grammars (a survey)”. In: *Graph-Grammars and Their Application to Computer Science and Biology*. Ed. by Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1979, pp. 1–69. ISBN: 978-3-540-35091-0. DOI: 10.1007/BFb0025714 (cit. on pp. 3, 23, 24, 47, 84).

[56] Hartmut Ehrig, Karsten Ehrig, Annegret Habel, and Karl-Heinz Pennemann. “Constraints and Application Conditions: From Graphs to High-Level Structures”. In: *Graph Transformations*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 287–303. ISBN: 978-3-540-30203-2. DOI: 10.1007/978-3-540-30203-2_21 (cit. on pp. 72, 136).

[57] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Berlin Heidelberg: Springer-Verlag, 2006. ISBN: 978-3-540-31187-4. DOI: 10.1007/3-540-31188-2 (cit. on pp. 3, 23, 31, 32, 35, 38, 48, 60, 71, 84, 129–131, 133, 182).

[58] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. “M-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation”. In: *Mathematical Structures in Computer Science* 24.4 (Aug. 2014). ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129512000357 (cit. on pp. 73, 75, 84).

[59] Hartmut Ehrig, Annegret Habel, Julia Padberg, and Ulrike Prange. “Adhesive High-Level Replacement Categories and Systems”. In: *Graph Transformations*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 144–160. DOI: 10.1007/978-3-540-30203-2_12 (cit. on p. 37).

[60] Hartmut Ehrig, Martin Korff, and Michael Löwe. “Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts”. In: *Graph Grammars and Their Application to Computer Science*. Ed. by Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 532. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1991, pp. 24–37. ISBN: 978-3-540-38395-6. DOI: 10.1007/BFb0017375 (cit. on pp. 23, 24, 41).

[61] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, eds. *Handbook of Graph Grammars and Computing by Graph Transformation: Concurrency, Parallelism, and Distribution*. Vol. 3. Concurrency, Parallelism, and Distribution. USA: World Scientific, 1999. ISBN: 978-981-4494-42-7. DOI: 10.1142/4181 (cit. on p. 42).

[62] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985. ISBN: 978-3-642-69964-1. DOI: 10.1007/978-3-642-69962-7 (cit. on pp. 126, 127).

[63] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. “Fundamental Theory for Typed Attributed Graph Transformation”. In: *Graph Transformations*. ICGT 2004. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Vol. 3256. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 161–177. ISBN: 978-3-540-30203-2. DOI: 10.1007/978-3-540-30203-2_13 (cit. on pp. 126, 129).

[64] Hartmut Ehrig and Barry K. Rosen. “Parallelism and concurrency of graph manipulations”. In: *Theoretical Computer Science* 11.3 (July 1, 1980), pp. 247–275. ISSN: 0304-3975. DOI: 10.1016/0304-3975(80)90016-X (cit. on p. 84).

[65] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. “WorldBrush: interactive example-based synthesis of procedural virtual worlds”. In: *ACM Transactions on Graphics* 34.4 (July 27, 2015), 106:1–106:11. ISSN: 0730-0301. DOI: 10.1145/2766975 (cit. on p. 208).

[66] Joost Engelfriet and Grzegorz Rozenberg. “Node replacement graph grammars”. In: *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. doi: 10.5555/278918.278925. USA: World Scientific, Feb. 1, 1997, pp. 1–94. ISBN: 978-981-02-2884-2 (cit. on p. 87).

[67] G. Farin, J. Hoschek, and M.-S. Kim. *Handbook of Computer Aided Geometric Design*. Elsevier, Aug. 13, 2002. 850 pp. ISBN: 978-0-444-51104-1 (cit. on p. 13).

[68] Shahjadi Hisan Farjana and Soonhung Han. “Mechanisms of Persistent Identification of Topological Entities in CAD Systems: A Review”. In: *Alexandria Engineering Journal* 57.4 (Dec. 1, 2018), pp. 2837–2849. ISSN: 1110-0168. DOI: 10.1016/j.aej.2018.01.007 (cit. on p. 208).

[69] Marek Fiser, Bedrich Benes, Jorge Garcia Galicia, Michel Abdul-Massih, Daniel G. Aliaga, and Vojtech Krs. “Learning geometric graph grammars”. In: *Proceedings of the 32nd Spring Conference on Computer Graphics*. SCCG ’16. Slomenice, Slovakia: Association for Computing Machinery, Apr. 27, 2016, pp. 7–15. ISBN: 978-1-4503-4436-4. DOI: 10.1145/2948628.2948635 (cit. on p. 43).

[70] Scott Fortin. *The Graph Isomorphism Problem*. University of Alberta Libraries, 1996. DOI: 10.7939/R3SX64C5K (cit. on p. 186).

[71] Blender Foundation. *Blender*. URL: https://www.blender.org/ (visited on 08/24/2022) (cit. on p. 14).

[72] Thomas M. J. Fruchterman and Edward M. Reingold. “Graph drawing by force-directed placement”. In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. ISSN: 1097-024X. DOI: 10.1002/spe.4380211102 (cit. on p. 223).

[73] Ignacio Garcia-Dorado, Daniel G. Aliaga, Saiprasanth Bhalachandran, Paul Schmid, and Dev Niyogi. “Fast Weather Simulation for Inverse Procedural Design of 3D Urban Models”. In: *ACM Transactions on Graphics* 36.2 (Apr. 7, 2017), 21:1–21:19. ISSN: 0730-0301. DOI: 10.1145/2999534 (cit. on p. 208).

[74] Valentin Gauthier. “Développement d’un langage de programmation dédié à la modélisation géométrique à base topologique, application à la reconstruction de modèles géologiques 3D”. These de doctorat. Poitiers, Jan. 17, 2019 (cit. on pp. 2, 4, 199–201, 230, 255).

[75] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam Szalkowski. "GrGen: A Fast SPO-Based Graph Rewriting Tool". In: *Graph Transformations*. ICGT 2006. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 383–397. ISBN: 978-3-540-38872-2. DOI: 10.1007/11841883_27 (cit. on p. 42).

[76] Jean-Louis Giavitto and Olivier Michel. "MGS: A Rule-Based Programming Language for Complex Objects and Collections". In: *Electronic Notes in Theoretical Computer Science*. RULE 2001, Second International Workshop on Rule-Based Programming (Satellite Event of PLI 2001) 59.4 (Nov. 1, 2001), pp. 286–304. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(04)00293-2 (cit. on p. 43).

[77] Helen Gibson, Joe Faith, and Paul Vickers. "A survey of two-dimensional graph layout techniques for information visualisation". In: *Information Visualization* 12.3 (July 1, 2013). Publisher: SAGE Publications, pp. 324–357. ISSN: 1473-8716. DOI: 10.1177/1473871612455749 (cit. on p. 223).

[78] Google. *OR-Tools*. URL: https://developers.google.com/optimization (cit. on p. 251).

[79] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. "Efficient search of combinatorial maps using signatures". In: *Theoretical Computer Science*. Theoretical Computer Science Issues in Image Analysis and Processing 412.15 (Mar. 25, 2011), pp. 1392–1405. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2010.10.029 (cit. on pp. 187, 227).

[80] David Gould. *Complete Maya Programming: An Extensive Guide to MEL and C++ API*. First Edition. The Morgan Kaufmann Series in Computer Graphics. Elsevier, 2003. 532 pp. ISBN: 978-1-55860-835-1. DOI: 10.1016/B978-1-55860-835-1.X5000-9 (cit. on p. 207).

[81] Guilherme Grochau Azzi, Andrea Corradini, and Leila Ribeiro. "On the essence and initiality of conflicts in M-adhesive transformation systems". In: *Journal of Logical and Algebraic Methods in Programming* 109 (Dec. 1, 2019), p. 100482. ISSN: 2352-2208. DOI: 10.1016/j.jlamp.2019.100482 (cit. on pp. 37, 129).

[82] Leonidas Guibas and Jorge Stolfi. "Primitives for the manipulation of general subdivisions and the computation of Voronoi". In: *ACM Transactions on Graphics* 4.2 (Apr. 1, 1985), pp. 74–123. ISSN: 0730-0301. DOI: 10.1145/282918.282923 (cit. on p. 14).

[83] Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. "Inverse Procedural Modeling of Branching Structures by Inferring L-Systems". In: *ACM Transactions on Graphics* 39.5 (June 15, 2020), 155:1–155:13. ISSN: 0730-0301. DOI: 10.1145/3394105 (cit. on p. 208).

[84] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. "Deep Learning for 3D Point Clouds: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (Dec. 2021), pp. 4338–4364. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3005434 (cit. on p. 12).

[85] J. Guzmán, A. Lischke, and M. Neilan. "Exact sequences on Powell–Sabin splits". In: *Calcolo* 57.2 (Mar. 13, 2020), p. 13. ISSN: 1126-5434. DOI: 10.1007/s10092-020-00361-x (cit. on pp. 237, 238).

[86] Marc Gyssens, Jan Paredaens, Jan van den Bussche, and Dirk van Gucht. “A graph-oriented object database model”. In: *IEEE Transactions on Knowledge and Data Engineering* 6.4 (Aug. 1994). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 572–586. ISSN: 1558-2191. DOI: 10.1109/69.298174 (cit. on p. 42).

[87] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. “Graph Grammars with Negative Application Conditions”. In: *Fundamenta Informaticae* 26.3 (Dec. 1996), pp. 287–313. ISSN: 0169-2968. DOI: 10.3233/FI-1996-263404 (cit. on p. 72).

[88] Annegret Habel, Jürgen Müller, and Detlef Plump. “Double-pushout graph transformation revisited”. In: *Mathematical Structures in Computer Science* 11.5 (Oct. 1, 2001), pp. 637–688. ISSN: 0960-1295. DOI: 10.1017/S0960129501003425 (cit. on p. 41).

[89] Annegret Habel and Karl-Heinz Pennemann. “Nested Constraints and Application Conditions for High-Level Structures”. In: *Formal Methods in Software and Systems Modeling: Essays Dedicated to Hartmut Ehrig on the Occasion of His 60th Birthday*. Ed. by Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 293–308. ISBN: 978-3-540-31847-7. DOI: 10.1007/978-3-540-31847-7_17 (cit. on pp. 72, 75).

[90] Annegret Habel and Karl-Heinz Pennemann. “Satisfiability of High-Level Conditions”. In: *Graph Transformations*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 430–444. ISBN: 978-3-540-38872-2. DOI: 10.1007/11841883_30 (cit. on p. 75).

[91] Annegret Habel and Karl-Heinz Pennemann. “Correctness of high-level transformation systems relative to nested conditions”. In: *Mathematical Structures in Computer Science* 19.2 (Apr. 2009), pp. 245–296. ISSN: 1469-8072, 0960-1295. DOI: 10.1017/S0960129508007202 (cit. on pp. 51, 73, 75).

[92] Annegret Habel, Karl-Heinz Pennemann, and Arend Rensink. “Weakest Preconditions for High-Level Programs”. In: *Graph Transformations*. Ed. by Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 445–460. ISBN: 978-3-540-38872-2. DOI: 10.1007/11841883_31 (cit. on p. 75).

[93] Annegret Habel and Detlef Plump. “Relabelling in Graph Transformation”. In: *Graph Transformation (ICGT 2002)*. Ed. by Andrea Corradini, Hartmut Ehrig, Hans -Jörg Kreowski, and Grzegorz Rozenberg. Vol. 2505. Lecture Notes in Computer Science. Springer, 2002, pp. 135–147. DOI: 10.1007/3-540-45832-8_12 (cit. on pp. 47, 86).

[94] Martin Haeusler, Thomas Trojer, Johannes Kessler, Matthias Farwick, Emmanuel Nowakowski, and Ruth Breu. “ChronoSphere: a graph-based EMF model repository for IT landscape models”. In: *Software and Systems Modeling* 18.6 (Dec. 1, 2019). Publisher: Springer, pp. 3487–3526. ISSN: 1619-1374. DOI: 10.1007/s10270-019-00725-0 (cit. on p. 42).

[95] Alain Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002. 551 pp. ISBN: 0-521-79540-0. URL: https://pi.math.cornell.edu/~hatcher/AT/AT.pdf (cit. on pp. 13, 16).

[96] Huahai He and Ambuj K. Singh. “Graphs-at-a-time: query language and access methods for graph databases”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD ’08. New York, NY, USA: Association for Computing Machinery, June 9, 2008, pp. 405–418. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376660 (cit. on p. 42).

[97] Reiko Heckel. “Panel Discussion: Learning Graph Transformation Rules”. In: *GCM 2022 Graph Computation Models*. Graph Computation Models. 2022, pp. 100–100 (cit. on p. 3).

[98] Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. “Confluence of Typed Attributed Graph Transformation Systems”. In: *Graph Transformation (ICGT 2002)*. Ed. by Andrea Corradini, Hartmut Ehrig, Hans -Jörg Kreowski, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 161–176. ISBN: 978-3-540-45832-6. DOI: 10.1007/3-540-45832-8_14 (cit. on pp. 126, 129, 168).

[99] Reiko Heckel and Gabriele Taentzer. *Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-43915-6. DOI: 10.1007/978-3-030-43916-3 (cit. on pp. 3, 23, 41, 42, 46, 47, 63, 71–73, 128, 136, 138, 208).

[100] C. A. R. Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (Oct. 1, 1969), pp. 576–580, 583. ISSN: 0001-0782. DOI: 10.1145/363235.363259 (cit. on p. 76).

[101] Berthold Hoffmann. “Graph Transformation with Variables”. In: *Formal Methods in Software and Systems Modeling*. Ed. by Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer. Vol. 3393. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 101–115. ISBN: 978-3-540-31847-7. DOI: 10.1007/978-3-540-31847-7_6 (cit. on pp. 81, 84, 189).

[102] S. Horna, D. Meneveaux, G. Damiand, and Y. Bertrand. “Consistency constraints and 3D building reconstruction”. In: *Computer-Aided Design* 41.1 (Jan. 1, 2009), pp. 13–27. ISSN: 0010-4485. DOI: 10.1016/j.cad.2008.11.006 (cit. on p. 4).

[103] Yiwei Hu, Julie Dorsey, and Holly Rushmeier. “A novel framework for inverse procedural texture modeling”. In: *ACM Transactions on Graphics* 38.6 (Nov. 8, 2019), 186:1–186:14. ISSN: 0730-0301. DOI: 10.1145/3355089.3356516 (cit. on p. 208).

[104] Gérard Huet. *Initiation à la Théorie des Catégories*. Lecture notes, DEA "Fonctionalité, Structures de Calcul et Programmation", Paris VII. Dec. 20, 1987 (cit. on p. 23).

[105] Satoshi Ikehata, Hang Yang, and Yasutaka Furukawa. “Structured Indoor Modeling”. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE International Conference on Computer Vision. ICCV ’15. USA: IEEE Computer Society, 2015, pp. 1323–1331. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.156 (cit. on p. 43).

[106] Alec Jacobson, Daniele Panozzo, et al. *libigl: A simple C++ geometry processing library*. 2018. URL: https://libigl.github.io/ (cit. on p. 14).

[107] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software". In: *PLOS ONE* 9.6 (June 2014), pp. 1–12. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0098679 (cit. on p. 223).

[108] Tomihisa Kamada and Satoru Kawai. "An algorithm for drawing general undirected graphs". In: *Information processing letters* 31.1 (Apr. 12, 1989), p. 9. DOI: 10.1016/0020-0190(89)90102-6 (cit. on p. 223).

[109] Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. "UCSG-Net – Unsupervised Discovering of Constructive Solid Geometry Tree". In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). Oct. 20, 2020. DOI: 10.48550/arXiv.2006.09102 (cit. on pp. 1, 209).

[110] Leif Kobbelt. "sqrt(3)-subdivision". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., July 1, 2000, pp. 103–112. ISBN: 978-1-58113-208-3. DOI: 10.1145/344779.344835 (cit. on p. 238).

[111] Barbara König, Dennis Nolte, Julia Padberg, and Arend Rensink. "A Tutorial on Graph Transformation". In: *Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig*. Ed. by Reiko Heckel and Gabriele Taentzer. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 83–104. ISBN: 978-3-319-75396-6. DOI: 10.1007/978-3-319-75396-6_5 (cit. on pp. 23, 128).

[112] Pierre Kraemer, David Cazier, and Dominique Bechmann. "Extension of half-edges for the representation of multiresolution subdivision surfaces". In: *The Visual Computer* 25.2 (Feb. 1, 2009), pp. 149–163. DOI: 10.1007/s00371-008-0211-6 (cit. on p. 14).

[113] Pierre Kraemer, Lionel Untereiner, Thomas Jund, Sylvain Thery, and David Cazier. "CGoGN: n-dimensional Meshes with Combinatorial Maps". In: *Proceedings of the 22nd International Meshing Roundtable*. Ed. by Josep Sarrate and Matthew Staten. Cham: Springer International Publishing, 2014, pp. 485–503. ISBN: 978-3-319-02335-9. DOI: 10.1007/978-3-319-02335-9_27 (cit. on p. 6).

[114] Jiri Kripac. "A mechanism for persistently naming topological entities in history-based parametric solid models". In: *Proceedings of the third ACM symposium on Solid modeling and applications*. SMA '95. New York, NY, USA: Association for Computing Machinery, Dec. 1, 1995, pp. 21–30. ISBN: 978-0-89791-672-1. DOI: 10.1145/218013.218024 (cit. on p. 208).

[115] Stephen Lack and Paweł Sobociński. "Adhesive Categories". In: *Foundations of Software Science and Computation Structures*. Ed. by Igor Walukiewicz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 273–288. ISBN: 978-3-540-24727-2. DOI: 10.1007/978-3-540-24727-2_20 (cit. on pp. 37, 38, 60, 151).

[116] Stephen Lack and Paweł Sobociński. "Adhesive and quasiadhesive categories". In: *RAIRO - Theoretical Informatics and Applications* 39.3 (July 1, 2005), pp. 511–545. ISSN: 0988-3754, 1290-385X. DOI: 10.1051/ita:2005028 (cit. on p. 37).

[117] Sergei K. Lando and Alexander K. Zvonkin. “Constellations, Coverings, and Maps”. In: *Graphs on Surfaces and Their Applications*. Ed. by Sergei K. Lando and Alexander K. Zvonkin. Encyclopaedia of Mathematical Sciences. Berlin, Heidelberg: Springer, 2004, pp. 7–77. ISBN: 978-3-540-38361-1. DOI: 10.1007/978-3-540-38361-1_2 (cit. on p. 15).

[118] Tom Leinster. *Basic Category Theory*. Dec. 29, 2016. arXiv: 1612.09375. URL: http://arxiv.org/abs/1612.09375 (cit. on pp. 23–30).

[119] Pascal Lienhardt. “Subdivisions of N-dimensional Spaces and N-dimensional Generalized Maps”. In: *Proceedings of the Fifth Annual Symposium on Computational Geometry*. SCG ’89. New York, NY, USA: Association for Computing Machinery, June 5, 1989, pp. 228–236. ISBN: 978-0-89791-318-8. DOI: 10.1145/73833.73859 (cit. on pp. 4, 15).

[120] Pascal Lienhardt. “Topological models for boundary representation: a comparison with n-dimensional generalized maps”. In: *Computer-Aided Design* 23.11 (Feb. 1991), pp. 59–82. ISSN: 0010-4485. DOI: 10.1016/0010-4485(91)90100-B (cit. on pp. 4, 14).

[121] Pascal Lienhardt. “N-dimensional generalized combinatorial maps and cellular quasi-manifolds”. In: *International Journal of Computational Geometry & Applications* 4.3 (Sept. 1, 1994), pp. 275–324. ISSN: 0218-1959. DOI: 10.1142/S0218195994000173. (Visited on 08/19/2020) (cit. on p. 16).

[122] Aristid Lindenmayer. “Developmental systems without cellular interactions, their languages and grammars”. In: *Journal of Theoretical Biology* 30.3 (1971), pp. 455–484. DOI: 10.1016/0022-5193(71)90002-6 (cit. on p. 18).

[123] Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. “Neural subdivision”. In: *ACM Transactions on Graphics* 39.4 (July 8, 2020), 124:124:1–124:124:16. ISSN: 0730-0301. DOI: 10.1145/3386569.3392418 (cit. on pp. 1, 208).

[124] Charles Loop. “Smooth Subdivision Surfaces Based on Triangles”. Masters Thesis. Department of Mathematics, University of Utah, Aug. 1987. 74 pp. URL: http://charlesloop.com/thesis.pdf (cit. on pp. 208, 236).

[125] Jesús J. López-Fernández, Antonio Garmendia, Esther Guerra, and Juan de Lara. “An example is worth a thousand words: Creating graphical modelling environments by example”. In: *Software & Systems Modeling* 18.2 (2019). Publisher: Springer, pp. 961–993. DOI: 10.1007/s10270-017-0632-7 (cit. on p. 209).

[126] Eugene M. Luks. “Isomorphism of graphs of bounded valence can be tested in polynomial time”. In: *Journal of Computer and System Sciences* 25.1 (Aug. 1, 1982), pp. 42–65. ISSN: 0022-0000. DOI: 10.1016/0022-0000(82)90009-5 (cit. on p. 187).

[127] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics*. Google-Books-ID: 0WbmCgAAQBAJ. CRC Press, Nov. 18, 2015. 737 pp. ISBN: 978-1-4822-2941-7 (cit. on p. 12).

[128] Brendan D. McKay and Adolfo Piperno. “Practical graph isomorphism, II”. In: *Journal of Symbolic Computation* 60 (Jan. 1, 2014), pp. 94–112. ISSN: 0747-7171. DOI: 10.1016/j.jsc.2013.09.003 (cit. on pp. 186, 227).

[129] Donald J. Miller. “The Categorical Product of Graphs”. In: *Canadian Journal of Mathematics* 20 (1968). Publisher: Cambridge University Press, pp. 1511–1521. ISSN: 0008-414X, 1496-4279. DOI: 10.4153/CJM-1968-151-x (cit. on p. 33).

[130] M. Minas and G. Viehstaedt. “DiaGen: a generator for diagram editors providing direct manipulation and execution of diagrams”. In: *Proceedings of Symposium on Visual Languages*. Proceedings of Symposium on Visual Languages. ISSN: 1049-2615. Sept. 1995, pp. 203–210. DOI: 10.1109/VL.1995.520810 (cit. on p. 42).

[131] Leonardo de Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 337–340. ISBN: 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24 (cit. on p. 251).

[132] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. “Procedural modeling of buildings”. In: *ACM SIGGRAPH 2006*. SIGGRAPH ’06. New York, NY, USA: Association for Computing Machinery, July 1, 2006, pp. 614–623. ISBN: 978-1-59593-364-5. DOI: 10.1145/1179352.1141931 (cit. on p. 18).

[133] Ulrich Nickel, Jörg Niere, and Albert Zündorf. “The FUJABA environment”. In: *Proceedings of the 22nd international conference on Software engineering*. ICSE ’00. New York, NY, USA: Association for Computing Machinery, June 1, 2000, pp. 742–745. ISBN: 978-1-58113-206-9. DOI: 10.1145/337180.337620 (cit. on p. 42).

[134] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. “Sketch-based modeling: A survey”. In: *Computers & Graphics* 33.1 (Feb. 1, 2009), pp. 85–103. ISSN: 0097-8493. DOI: 10.1016/j.cag.2008.09.013 (cit. on p. 269).

[135] Yongzhi Ong and Winfried Kurth. “A graph model and grammar for multi-scale modelling using XL”. In: *2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops*. 2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops. IEEE, Oct. 2012, pp. 1–8. DOI: 10.1109/BIBMW.2012.6470293 (cit. on p. 43).

[136] Yoav I. H. Parish and Pascal Müller. “Procedural modeling of cities”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’01. New York, NY, USA: Association for Computing Machinery, Aug. 1, 2001, pp. 301–308. ISBN: 978-1-58113-374-5. DOI: 10.1145/383259.383292 (cit. on p. 18).

[137] Romain Pascual, Hakim Belhaouari, Agnès Arnould, and Pascale Le Gall. “A First Step Towards the Inference of Geological Topological Operations”. In: *Eurographics 2022 - Posters*. Eurographics 2022. Ed. by Basile Sauvage and Jasminka Hasic-Telalovic. The Eurographics Association, 2022, p. 2. ISBN: 978-3-03868-171-7. DOI: 10.2312/egp.20221005 (cit. on p. 230).

[138] Romain Pascual, Hakim Belhaouari, Agnès Arnould, and Pascale Le Gall. “Inferring topological operations on generalized maps: Application to subdivision schemes”. In: *Graphics and Visual Computing* 6 (May 14, 2022), p. 200049. ISSN: 2666-6294. DOI: 10.1016/j.gvc.2022.200049 (cit. on pp. 9, 189, 190, 235, 301).

[139] Romain Pascual, Pascale Le Gall, Agnès Arnould, and Hakim Belhaouari. “Topological consistency preservation with graph transformation schemes”. In: *Science of Computer Programming* 214 (Feb. 1, 2022), p. 102728. ISSN: 0167-6423. DOI: 10.1016/j.scico.2021.102728 (cit. on pp. 8, 47, 64, 71, 78, 81, 102, 104, 108, 109, 114, 158, 293).

[140] Karl-Heinz Pennemann. “Development of correct graph transformation systems”. PhD thesis. Department für Informatik, Universität Oldenburg, 2009. URL: http://oops.uni-oldenburg.de/884 (cit. on p. 76).

[141] Detlef Plump. “The Graph Programming Language GP”. In: *Algebraic Informatics*. Ed. by Symeon Bozapalidis and George Rahonis. Vol. 5725. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 99–122. ISBN: 978-3-642-03564-7. DOI: 10.1007/978-3-642-03564-7_6 (cit. on pp. 76, 129, 133).

[142] Christopher M. Poskitt and Detlef Plump. “Hoare-Style Verification of Graph Programs”. In: *Fundamenta Informaticae* 118.1 (Jan. 1, 2012), pp. 135–175. ISSN: 0169-2968. DOI: 10.3233/FI-2012-708 (cit. on p. 76).

[143] Mathieu Poudret. “Transformations de graphes pour les opérations topologiques en modélisation géométrique : application à l’étude de la dynamique de l’appareil de Golgi”. PhD thesis. Evry-Val d’Essonne, Oct. 15, 2009 (cit. on p. 17).

[144] Mathieu Poudret, Agnès Arnould, Jean-Paul Comet, and Pascale Le Gall. “Graph Transformation for Topology Modelling”. In: *Graph Transformations*. ICGT 2008. Ed. by Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 147–161. DOI: 10.1007/978-3-540-87405-8_11 (cit. on pp. 37, 43, 47, 49, 50, 54, 56, 58–61, 81, 84, 125, 143, 158, 189).

[145] Mathieu Poudret, Jean-Paul Comet, Pascale Le Gall, Agnès Arnould, and Philippe Meseure. “Topology-based Geometric Modelling for Biological Cellular Processes”. In: *International Conference on Language and Automata Theory and Applications*. Tarragone, Spain, Mar. 2007, pp. 497–508 (cit. on pp. 43, 56, 125, 189).

[146] Michael James David Powell and Malcolm Sabin. “Piecewise Quadratic Approximations on Triangles”. In: *ACM Transactions on Mathematical Software* 3.4 (Dec. 1977), pp. 316–325. ISSN: 0098-3500, 1557-7295. DOI: 10.1145/355759.355761 (cit. on p. 237).

[147] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. “Graphical modeling using L-systems”. In: *The Algorithmic Beauty of Plants*. The Virtual Laboratory. New York, NY: Springer, 1990, pp. 1–50. DOI: 10.1007/978-1-4613-8476-2_1 (cit. on p. 18).

[148] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. “Development models of herbaceous plants for computer imagery purposes”. In: *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’88. New York, NY, USA: Association for Computing Machinery, June 1, 1988, pp. 141–150. ISBN: 978-0-89791-275-4. DOI: 10.1145/54852.378503 (cit. on p. 18).

[149] Przemyslaw Prusinkiewicz, Faramarz Samavati, Colin Smith, and Radoslaw Karwowski. “L-system description of subdivision curves”. In: *International Journal of Shape Modeling* 09.1 (June 1, 2003). Publisher: World Scientific Publishing Co., pp. 41–59. ISSN: 0218-6543. DOI: 10.1142/S0218654303000048 (cit. on p. 208).

[150] François Puitg and Jean -François Dufourd. “Formal specification and theorem proving breakthroughs in geometric modeling”. In: *Theorem Proving in Higher Order Logics*. Ed. by Jim Grundy and Malcolm Newey. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 401–422. ISBN: 978-3-540-49801-8. DOI: 10.1007/BFb0055149 (cit. on p. 19).

[151] Arend Rensink. “Representing First-Order Logic Using Graphs”. In: *Graph Transformations*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 319–335. ISBN: 978-3-540-30203-2. DOI: 10.1007/978-3-540-30203-2_23 (cit. on p. 75).

[152] Arend Rensink. “The GROOVE Simulator: A Tool for State Space Generation”. In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by John L. Pfaltz, Manfred Nagl, and Boris Böhlen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 479–485. ISBN: 978-3-540-25959-6. DOI: 10.1007/978-3-540-25959-6_40 (cit. on p. 42).

[153] Arend Rensink and Jan-Hendrik Kuperus. “Repotting the Geraniums: On Nested Graph Transformation Rules”. In: *Electronic Communications of the EASST* 18.0 (Sept. 8, 2009). ISSN: 1863-2122. DOI: 10.14279/tuj.eceasst.18.260 (cit. on p. 84).

[154] Lydie Richaume, Eric Andres, Gaëlle Largeteau-Skapin, and Rita Zrour. “Unfolding Level 1 Menger Polycubes of Arbitrary Size With Help of Outer Faces”. In: *Discrete Geometry for Computer Imagery*. Ed. by Michel Couprie, Jean Cousty, Yukiko Kenmochi, and Nabil Mustafa. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 457–468. ISBN: 978-3-030-14085-4. DOI: 10.1007/978-3-030-14085-4_36 (cit. on pp. 262, 267).

[155] Marko A. Rodriguez. “The Gremlin graph traversal machine and language (invited talk)”. In: *Proceedings of the 15th Symposium on Database Programming Languages*. DBPL 2015. New York, NY, USA: Association for Computing Machinery, Oct. 27, 2015, pp. 1–10. ISBN: 978-1-4503-3902-5. DOI: 10.1145/2815072.2815073. (Visited on 08/28/2020) (cit. on pp. 42, 168).

[156] Grzegorz Rozenberg, ed. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations*. Vol. Foundations. 1 vols. USA: World Scientific Publishing Co., Inc., Feb. 1, 1997. 545 pp. ISBN: 978-981-02-2884-2 (cit. on p. 182).

[157] Grzegorz Rozenberg and Arto Salomaa. *The Mathematical Theory of L Systems*. Academic press, 1980. 351 pp. ISBN: 978-0-08-087406-7 (cit. on p. 18).

[158] Edmar Santos and Regina Celia Coelho. “Obtaining L-Systems Rules from Strings”. In: *2009 3rd Southern Conference on Computational Modeling*. 2009 3rd Southern Conference on Computational Modeling. Nov. 2009, pp. 143–149. DOI: 10.1109/MCSUL.2009.21 (cit. on p. 208).

[159] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605 (cit. on p. 205).

[160] Andy Schürr, Andreas J. Winter, and Albert Zündorf. “Graph grammar engineering with PROGRES”. In: *European Software Engineering Conference*. Ed. by Wilhelm Schäfer and Pere Botella. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 219–234. ISBN: 978-3-540-45552-3. DOI: 10.1007/3-540-60406-5_17 (cit. on p. 42).

[161] Jean-Pierre Serre. *Trees*. Springer, 1980. ISBN: 978-3-642-61856-7 (cit. on p. 34).

[162] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. “CSGNet: Neural Shape Parser for Constructive Solid Geometry”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Mar. 31, 2018, pp. 5515–5523. DOI: 10.48550/arXiv.1712.08290. arXiv: 1712.08290 (cit. on pp. 1, 209).

[163] Nicholas Sharp and Keenan Crane. *Geometry Central*. 2019. URL: www.geometry-central.net (cit. on p. 14).

[164] Ruben M Smelik, Klaas Jan de Kraker, Saskia A Groenewegen, Tim Tutenel, and Rafael Bidarra. “A Survey of Procedural Methods for Terrain Modelling”. In: *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*. CASA Workshop in 3D advanced media in gaming and simulation (3AMIGAS). 2009, pp. 25–34. ISBN: 978-90-393-5102-4 (cit. on p. 18).

[165] Colin Smith, Przemyslaw Prusinkiewicz, and Faramarz Samavati. “Local Specification of Surface Subdivision Algorithms”. In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by John L. Pfaltz, Manfred Nagl, and Boris Böhlen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 313–327. ISBN: 978-3-540-25959-6. DOI: 10.1007/978-3-540-25959-6_23 (cit. on p. 13).

[166] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. “Algorithmic Self-assembly by Accretion and by Carving in MGS”. In: *Artificial Evolution*. Ed. by El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 189–200. ISBN: 978-3-540-33590-0. DOI: 10.1007/11740698_17 (cit. on p. 43).

[167] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. “Declarative Mesh Subdivision Using Topological Rewriting in MGS”. In: *Graph Transformations*. Ed. by Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 298–313. ISBN: 978-3-642-15928-2. DOI: 10.1007/978-3-642-15928-2_20 (cit. on p. 43).

[168] Amy Henderson Squillacote, James Ahrens, Charles Law, Berk Geveci, Kenneth Moreland, and Brad King. *The ParaView guide*. Clifton Park, NY: Kitware, 2007. 366 pp. ISBN: 978-1-930934-21-4 (cit. on p. 207).

[169] O. Šťava, B. Beneš, R. Měch, D. G. Aliaga, and P. Krištof. “Inverse Procedural Modeling by Automatic Generation of L-systems”. In: *Computer Graphics Forum* 29.2 (2010), pp. 665–674. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2009.01636.x. (Visited on 02/10/2022) (cit. on pp. 4, 208).

[170] O. Šťava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Beneš. “Inverse Procedural Modelling of Trees”. In: *Computer Graphics Forum* 33.6 (2014), pp. 118–131. ISSN: 1467-8659. DOI: 10.1111/cgf.12282 (cit. on pp. 1, 207).

[171] Gabriele Taentzer. “AGG: A tool environment for algebraic graph transformation”. In: *International Workshop on Applications of Graph Transformations with Industrial Relevance*. Applications of Graph Transformations with Industrial Relevance (AGTIVE). Berlin Heidelberg: Springer, 1999, pp. 481–488. ISBN: 978-3-540-67658-4. DOI: 10.1007/3-540-45104-8_41 (cit. on p. 42).

[172] Gabriele Taentzer. “AGG: A Graph Transformation Environment for Modeling and Validation of Software”. In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by John L. Pfaltz, Manfred Nagl, and Boris Böhlen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 446–453. ISBN: 978-3-540-25959-6. DOI: 10.1007/978-3-540-25959-6_35 (cit. on p. 42).

[173] Gabriele Taentzer. “Instance Generation from Type Graphs with Arbitrary Multiplicities”. In: *Electronic Communications of the EASST* 47.0 (July 12, 2012). Number: 0. ISSN: 1863-2122. DOI: 10.14279/tuj.eceasst.47.727 (cit. on p. 136).

[174] Gabriele Taentzer and Martin Beyer. “Amalgamated graph transformations and their use for specifying AGG — an algebraic graph grammar system”. In: *Graph Transformations in Computer Science*. Ed. by Hans Jürgen Schneider and Hartmut Ehrig. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1994, pp. 380–394. ISBN: 978-3-540-48333-5. DOI: 10.1007/3-540-57787-4_24 (cit. on p. 84).

[175] Gabriele Taentzer and Arend Rensink. “Ensuring Structural Constraints in Graph-Based Models with Type Inheritance”. In: *Fundamental Approaches to Software Engineering*. Ed. by Maura Cerioli. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 64–79. ISBN: 978-3-540-31984-9. DOI: 10.1007/978-3-540-31984-9_6 (cit. on p. 136).

[176] Julien Tierny, Guillaume Favelier, Joshua A. Levine, Charles Gueunet, and Michael Michaux. “The Topology ToolKit”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 832–842. ISSN: 1941-0506. DOI: 10.1109/TVCG.2017.2743938 (cit. on p. 13).

[177] Simon Vilgertshofer and André Borrmann. “Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models”. In: *Advanced Engineering Informatics* 33 (Aug. 1, 2017). Publisher: Elsevier, pp. 502–515. ISSN: 1474-0346. DOI: 10.1016/j.aei.2017.07.003 (cit. on p. 43).

[178] Andrew Vince. “Combinatorial maps”. In: *Journal of Combinatorial Theory, Series B* 34.1 (Feb. 1, 1983), pp. 1–21. ISSN: 0095-8956. DOI: 10.1016/0095-8956(83)90002-3 (cit. on p. 14).

[179] Mitchell Wand. “Final algebra semantics and data type extensions”. In: *Journal of Computer and System Sciences* 19.1 (Aug. 1, 1979), pp. 27–44. ISSN: 0022-0000. DOI: 10.1016/0022-0000(79)90011-4 (cit. on p. 128).

[180] Joe Warren. *Subdivision methods for geometric design*. Nov. 1995 (cit. on p. 236).

[181] Jens H. Weber. “GRAPE – A Graph Rewriting and Persistence Engine”. In: *Graph Transformation*. Ed. by Juan de Lara and Detlef Plump. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 209–220. ISBN: 978-3-319-61470-0. DOI: 10.1007/978-3-319-61470-0_13 (cit. on p. 42).

[182] Jens H. Weber. “Tool Support for Functional Graph Rewriting with Persistent Data Structures - GrapeVine”. In: *Graph Transformation*. Ed. by Nicolas Behr and Daniel Strüber. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 195–206. ISBN: 978-3-031-09843-7. DOI: 10.1007/978-3-031-09843-7_11 (cit. on p. 42).

[183] Kevin J. Weiler. “Topological Structures for Geometric Modeling”. PhD thesis. New York, NY, USA: Rensselaer Polytechnic Institute, 1986. 340 pp. (cit. on p. 14).

[184] Martin Wirsing. “CHAPTER 13 - Algebraic Specification”. In: *Formal Models and Semantics*. Ed. by Jan van Leeuwen. Vol. B. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, 1990, pp. 675–788. ISBN: 978-0-444-88074-1. DOI: 10.1016/B978-0-444-88074-1.50018-4 (cit. on pp. 126, 127).

[185] Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. “Inverse procedural modeling of facade layouts”. In: *ACM Transactions on Graphics* 33.4 (July 27, 2014), 121:1–121:10. ISSN: 0730-0301. DOI: 10.1145/2601097.2601162 (cit. on p. 207).

[186] Jun Wu, Rüdiger Westermann, and Christian Dick. “Physically-based Simulation of Cuts in Deformable Bodies: A Survey”. In: *Eurographics (State of the Art Reports)* (2014). Ed. by Sylvain Lefebvre and Michela Spagnuolo, pp. 1–19. ISSN: 1017-4656. DOI: 10.2312/egst.20141033 (cit. on p. 4).

[187] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. “Inferring CAD Modeling Sequences Using Zone Graphs”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 6062–6070. DOI: 10.48550/arXiv.2104.03900. arXiv: 2104.03900 (cit. on p. 209).

[188] Kai Ching You and King-Sun Fu. “A Syntactic Approach to Shape Recognition Using Attributed Grammars”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.6 (June 1979), pp. 334–345. ISSN: 2168-2909. DOI: 10.1109/TSMC.1979.4310222 (cit. on p. 42).

Index

Appendices

Appendix A

Proofs of the results on the topology consistency

We provide the complete proofs of Chapter 3. Proofs are taken from [139].

A.1 Non-orientation condition

We prove Theorem 2 about the preservation of the non- orientation constraint with the non-orientation condition.

Let $r = \mathrm{L} \xleftarrow{i_\mathrm{L}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_\mathrm{R}} \mathrm{R}$ be a $\mathbb{D}$-combinatorial rule, and i a dimension in $\mathbb{D}$.

($\Longrightarrow$) We suppose r satisfies the non-orientation condition $\mathtt{O}_r(i)$ and show that for all matches $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{O}_\mathrm{G}(i)$, the graph H, result of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$, satisfies the non-orientation constraint $\mathtt{O}_\mathrm{H}(i)$.

Let $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ be a match with $\mathrm{G} \in \mathbb{D}$-**CGraph** and $\mathrm{G} \models \mathtt{O}_\mathrm{G}(i)$. Let H be the result of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ and m' be the comatch $\mathrm{R} \hookrightarrow \mathrm{H}$ Let v_H be a node in $\mathrm{V_H}$.

Suppose e_H is an i-arc incident to v_H. Let us show that e_H has a unique reverse i-arc e_H^{-1} in H.

1. If $e_\mathrm{H} \in \mathrm{H} \setminus m'(\mathrm{R})$ then $e_\mathrm{H} \in \mathrm{G} \setminus m(\mathrm{L})$. Thus, there exists $e_\mathrm{G} \in \mathrm{G}$ such that $e_\mathrm{G} = e_\mathrm{H}^{-1}$ in G.
   (a) If $e_\mathrm{G} \in \mathrm{G} \setminus m(\mathrm{L})$, both arcs are left unchanged by r, i.e., e_H and $e_\mathrm{G} = e_\mathrm{H}^{-1}$ are in H.
   (b) Otherwise, there exists e' in L such that $m(e') = e_\mathrm{G}$. Because G is a combinatorial graph and e_H is in $\mathrm{G} \setminus m(\mathrm{L})$, e' is oriented in L. Since r satisfies $\mathtt{O}_r(i)$, e' is in $\mathrm{L} \cap \mathrm{R}$. Therefore, $m'(e') = e_\mathrm{G} = e_\mathrm{H}^{-1}$ in H, by injectivity of m'.
2. Otherwise, $e_\mathrm{H} \in m'(\mathrm{R})$ and, by injectivity of m', there exists a unique $e \in \mathrm{R}$ such that $m'(e) = e_\mathrm{H}$.
   (a) If e admits a reverse arc e' in R, then $m'(e')$ is the reverse arc of e_H in H.
   (b) Otherwise, similarly to case 1b, e is an oriented arc of $\mathrm{L} \cap \mathrm{R}$. The arc e cannot have a reverse arc in L, as this arc would be in $\mathrm{E_L} \setminus \mathrm{E_R}$ and would break the non-orientation condition on r. Since e is in L, $m(e) = e_\mathrm{H}$. The non-orientation constraint in G yields an arc e_G reverse of e_H in G. This arc has no antecedent in L and belongs to $\mathrm{G} \setminus m(\mathrm{L})$. Therefore, e_G is in H where it is the reverse arc of e_H.

Thereafter H satisfies the non-orientation constraint $\mathtt{O}_{\mathrm{H}}(i)$.

(⟸) We now suppose that for all matches $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{O}_{\mathrm{G}}(i)$, the graph H, result of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$, satisfies the non-orientation constraint $\mathtt{O}_{\mathrm{H}}(i)$ and show that r satisfies the non-orientation condition $\mathtt{O}_r(i)$.

Let m' be the comatch $\mathrm{R} \hookrightarrow \mathrm{H}$, which is a mono by the construction of the double-pushout diagram, and consider an added arc e. Then, there exists e_{H} in H such that $m'(e) = e_{\mathrm{H}}$. Because H satisfies $\mathtt{O}_{\mathrm{H}}(i)$, e_{H} has a reverse arc e'_{H} in H. Since e is not in L, e_{H} is not in G. Because G satisfies $\mathtt{O}_{\mathrm{G}}(i)$, e'_{H} is not in G (otherwise e_{H} would also be in G). Thus, e'_{H} is an added arc in H. By injectivity of m' there exists a unique i-arcs e' in R such that $m'(e') = e'_{\mathrm{H}}$. Then, e and e' are reversed arcs of $\mathrm{E_R} \setminus \mathrm{E_L}$ in R. The proof holds by reversing the roles of H and G, as well as R and L.

Thereafter r satisfies the non-orientation condition $\mathtt{O}_r(i)$. □

A.2 Cycle condition

We prove Theorem 3 about the preservation of the cycle constraint with the cycle condition.

Let $r = \mathrm{L} \xleftarrow{i_{\mathrm{L}}} \mathrm{L} \cap \mathrm{R} \xhookrightarrow{i_{\mathrm{R}}} \mathrm{R}$ be a $\mathbb{D}$-combinatorial rule, and i and j be two dimensions in $\mathbb{D}$. Given a match $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{C}_{\mathrm{G}}(i,j)$, let m' denote the morphism $\mathrm{R} \hookrightarrow \mathrm{H}$, which is a mono by construction.

(⟹) Suppose r satisfies the cycle condition $\mathtt{C}_r(i,j)$.

Consider $m\colon \mathrm{L} \hookrightarrow \mathrm{G}$ a match on a $\mathbb{D}$-combinatorial graph G satisfying $\mathtt{C}_{\mathrm{G}}(i,j)$, and H the result of the direct derivation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$. Let v_{H} be a node of H.

▶ *Unmatched node*

Suppose that $v_{\mathrm{H}} \in \mathrm{H} \setminus m'(\mathrm{R})$, then v_{H} is a node of G. Thus, v_{H} is the source of an $ijij$-cycle in G since G satisfies $\mathtt{C}_{\mathrm{G}}(i,j)$.

1. If no arc of this cycle is in $m(\mathrm{L} \setminus \mathrm{R})$, then this cycle is left unchanged by the rule and v_{H} is the source of the same $ijij$-cycle in H.
2. Otherwise, certain parts of the cycle are in $m(\mathrm{L} \setminus \mathrm{R})$. Because v_{H} is in $\mathrm{G} \setminus m(\mathrm{L})$, two nodes $m(v_s)$ and $m(v_t)$ satisfies :
   - Nodes $m(v_s)$ and $m(v_t)$ belong to the cycle.
   - Nodes $m(v_s)$ and $m(v_t)$ are in $m(\mathrm{L} \cap \mathrm{R})$.
   - All arcs from the cycle in $p = m(v_s) \overset{w}{\rightsquigarrow} m(v_t)$ are in $\mathrm{G} \setminus m(\mathrm{L})$.
   - Node v_{H} belongs to p.

   Since v_{H} is in $\mathrm{G} \setminus m(\mathrm{L})$, its incident arcs cannot be in $m(\mathrm{L})$ and p is a path, the size of which is at least 2. If w' completes w to get a cycle, the remaining part of the cycle is a path $m(v_t) \overset{w'}{\rightsquigarrow} m(v_s)$, which is the image of an (i,j)-alternating path in L. This (i,j)-alternating path consists of interface arcs and optimal paths. Since any optimal path is strongly coherent according to sub-condition 1, there is a unique similar path in $m'(\mathrm{R})$. Either way, their concatenation with $m'(v_s) \overset{w}{\rightsquigarrow} m'(v_t)$ is an $ijij$-cycle of source v_{H} in H.

▶ *Matched node*

Otherwise, v_H is in $m'(R)$. Denote v the node in R such that $m'(v) = v_H$.

1. Suppose v is a node of $L \cap R$ and v is not the source of an i-arc in L. Since v belongs to $L \cap R$, v_H is a node of G. As G satisfies $\mathtt{C}_G(i,j)$, v_H is the source of an $ijij$-cycle in G. From that point, the proof is similar to cases 1 and 2 of unmatched nodes depending on whether the cycle is partially matched.

2. Otherwise, assume v is a node of $L \cap R$, but v is the source of an i-arc in L. Since r is a combinatorial rule, v is the source of an i-arc in R. Sub-condition 2 guarantees that this i-arc either belongs to an $ijij$-cycle or to an (i,j)-optimal path in R. In the former case, the image by m' of the cycle yields an $ijij$-cycle of source v_H in H. In the latter, let $p = v_s \overset{w}{\rightsquigarrow} v_t$ be the (i,j)-optimal path in R. By sub-condition 1, p is strongly coherent and there exists a unique (i,j)-optimal path $p' = v_s \overset{w}{\rightsquigarrow} v_t$ in L. Without loss of generality, assume that the first arc of p' is labeled by i. Since G satisfies $\mathtt{C}_G(i,j)$, $m(v_s)$ is the source of an $ijij$-cycle in G and the first part of this cycle is $m(p')$. The remaining arcs of the cycle can be subdivided into elements of $G \setminus m(L)$ left unchanged by the rule, interface arcs of $m(L \cap R)$ preserved by the rule, and (i,j)-optimal paths. Any such optimal path is strongly coherent from sub-condition 1 and yields a similar optimal path in R. Their concatenation with $m'(p)$ is a cycle. When considering it with v_H instead of $m(v_s)$, it is an $ijij$-cycle of source v_H in H.

3. If v is a node of $R \setminus L$, the proof is similar to case 2, exploiting sub-condition 3 instead of sub-condition 2.

Thereafter H satisfies the cycle constraint $\mathtt{C}_H(i,j)$.

($\Longleftarrow$) Suppose for all matches $m\colon L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathtt{C}_G(i,j)$, the result H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the cycle constraint $\mathtt{C}_H(i,j)$.

▶ *Sub-condition 1 of the cycle condition*

Suppose there is an (i,j)-optimal path $v_s \overset{w}{\rightsquigarrow} v_t$ in L. By optimality, $v_s \neq v_t$. According to Lemma 8, the path is of size 2. The path $m(v_s \overset{w}{\rightsquigarrow} v_t)$ belongs to an $ijij$-cycle in G because G satisfies $\mathtt{C}_G(i,j)$. The two arcs that define the path $m(v_t) \overset{w}{\rightsquigarrow} m(v_s)$ are in $G \setminus m(L \setminus R)$. Thus, they are in H, where they belong to an $ijij$-cycle. The incident arcs condition guarantees that the path $m'(v_t) \overset{w}{\rightsquigarrow} m'(v_s)$ comes entirely from $R \setminus L$ and is a maximal non-overlapping path. Since the path contains two arcs and has distinct endpoints, it cannot overlap. Therefore, it is the (i,j)-optimal path in R coherent with $v_s \overset{w}{\rightsquigarrow} v_t$. The incident arcs condition of combinatorial rules ensures the uniqueness of the optimal paths, which are therefore strongly coherent.

Suppose there is an (i,j)-optimal path $p = v_s \overset{w}{\rightsquigarrow} v_t$ in R and consider the cycle completion of the path in H. It yields elements in G that belong to a cycle with arcs or vertices from $m(L)$. Elements on this alternating path are in $m(L \setminus R)$ by hypothesis on p. According to Lemma 8, the path is of size 2. The incident arcs constraint on G ensures that it is the unique optimal path coherent with p.

► *Sub-condition 2 of the cycle condition*

Consider a preserved node v of $\mathrm{L} \cap \mathrm{R}$ that is the source of an i-arc in $\mathrm{L} \setminus \mathrm{R}$. Because r is a combinatorial rule, sub-condition 1 of the weak incident arcs condition $\mathtt{I}_{\mathrm{G}}(i)$ guarantees that v is also the source i-arc in $\mathrm{R} \setminus \mathrm{L}$ (this arc can not be in $\mathrm{L} \cap \mathrm{R}$ because v is already the source of an i-arc in $\mathrm{L} \setminus \mathrm{R}$).

Let v_{H} be the image of v by m', i.e., $v_{\mathrm{H}} = m'(v)$. Since H satisfies $\mathtt{C}_{\mathrm{H}}(i, j)$, v_{H} is the source of an $ijij$-cycle in H. Depending on whether the whole cycle is present in R, v is the source of an i-arc in R that belongs either to an $ijij$-cycle or to an (i, j)-optimal path. The proof holds for a j-arc.

► *Sub-condition 3 of the cycle condition*

Consider an added node v of $\mathrm{R} \setminus \mathrm{L}$. Because r is a combinatorial rule, sub-condition 2 of the weak incident arcs condition $\mathtt{I}_{\mathrm{G}}(i)$ guarantees that v is the source i-arc in R. Similar to the proof of sub-condition 2, this arc belongs either to an $ijij$-cycle or to an (i, j)-optimal path. The proof holds for a j-arc.

Thereafter r satisfies the cycle condition $\mathtt{C}_r(i, j)$. □

Appendix B

Proofs of the results on the embedding consistency

We provide the complete proofs of Chapter 5. Proofs are revisited versions of [3] to account for the construction of embeddings as node attributions.

B.1 Embedded rules

We prove Theorem 8 that ensures the preservation of the embedding constraint. We consider a set of embeddings Π, an embedding signature $\Omega(\Pi)$ for Π (Definition 68), an embedding algebra $\mathscr{A}(\Pi)$ (Definition 68), a Π- embedded combinatorial rule $r = \mathrm{L} \hookleftarrow \mathrm{R}$ (Definition 75) and an embedded match $m\colon \mathrm{L} \to \mathrm{G}$ (Definition 73) where G is an embedded combinatorial graph (Definition 72).

Since a combinatorial rule transforms a combinatorial graph into a combinatorial graph, we simply have to prove that the result H of the direct transformation $\mathrm{G} \Rightarrow^{r,m} \mathrm{H}$ is an embedded graph.

First, let us point out that H exists since the conditions of embedding consistency preservation and the definition of combinatorial rules imply the gluing condition for embedded graphs (Theorem 7). Let us also remark that the definition of embedded match ensures injectivity on the node, arc, and attribution arc functions. This injectivity is preserved by the pushouts resulting in the injectivity of the node, arc, and attribution arc functions of the comatch.

For the proof, we consider the following double-pushout diagram:

$$\begin{array}{ccccc} \mathrm{L} & \hookleftarrow & \mathrm{L}\cap\mathrm{R} & \hookrightarrow & \mathrm{R} \\ {\scriptstyle m}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle m'} \\ \mathrm{G} & \hookleftarrow & \mathrm{D} & \hookrightarrow & \mathrm{H} \end{array}$$

▶ *Unique existence of embedding*

We first prove the constraint of the unique existence of embedding, stating that each node should be the source of a unique attribution arc per embedding.

Let v be a node in $\mathrm{V_H}$ and π an embedding of Π. If v is not the image of some node v_R in R by m', then v belongs to G where it is the source of a π-attribution arc. This arc is not in the match and, therefore, in H. Otherwise, the condition of embedded components (Definition 74) ensures that R is an embedded graph. Therefore, the node v_R in R, whose image by m' is v is the source of a π-attribution arc in R. The comatch yields an π-attribution arc of source v in H. Note that if v

is a node of G, its π-attribution arc in G belongs to the match because L is an attributed graph. In particular, no other π-attribution arc of source v exists in H. By disjunction of cases, we obtain the constraint of the unique existence of embedding in H.

▶ *Orbit consistency*

We now prove the constraint of orbit consistency, stating that all nodes within the same embedding equivalence class should have the same embedding value.

Let π be an embedding of Π. Since we already proved the constraint of the unique existence of embedding, the orbit consistency states that for nodes v and w in H, $v \equiv_\pi w$ in H implies $\pi_H(v) = \pi_H(w)$. Moreover, because $\equiv_\pi$ is the reflexive, symmetric, and transitive closure of $\sim_\pi$, we can reason with $\sim_\pi$ instead of $\equiv_\pi$. In other words it suffices to show that for any i-arc e in H with $i \in \langle o_\pi \rangle$, $\pi_H(s_H(e)) = \pi_H(t_H(e))$.

Consider an i-arc e in H with i in $\langle o_\pi \rangle$, and let v be the source of e and w its target. We will write e', v', and w' for the element of R that respectively have e, v or w as image by m', when it exists. Similar to the proof of the constraint of the unique existence of embedding, if these elements exist, they are unique by injectivity of the comatch on the nodes and arcs.

1. If an arc e' exists in R such that $m'(e') = e$, then nodes v' and w' such that $m'(v') = v$ and $m'(w') = w$ also exist in R. Since R, is an embedded graph, the constraint of orbit consistency (Definition 71) ensures that $\pi_R(v') = \pi_R(w')$. Thus, $\pi_H(v) = \pi_H(w)$.

2. Otherwise, e is an arc of G.

   - If a node v' or a node w' exits in R such that $m'(v') = v$ or $m'(w') = w$, then it is a node of L∩R. Thus, the condition of the full orbits of transformed embeddings ensures that the associated embedding term is the same in both L and R (by contraposition). Furthermore, the algebra morphism is identical for the match and the comatch since the algebra morphisms in the rule are identities. Therefore, the values computed from the identical terms are equal in both G and H.
   - Otherwise, both nodes are not in the occurrence of the comatch. In such a case, these nodes correspond to nodes of G outside the occurrence of the match. In such a case, their embedding values do not change through derivation. Therefore, $\pi_H(v) = \pi_G(v) = \pi_G(w) = \pi_H(w)$.

Thereafter H is a π-embedded combinatorial graph. □

B.2 Embedding consistency preservation of the topological extension

We prove Theorem 11 that describes the embedding properties of the topological extension (Definition 78) of a weakly embedded combinatorial rule (Definition 74) along an embedded match (Definition 73) to an embedded combinatorial graph (Definition 72).

Let $r = L \xhookleftarrow{i_L} L \cap R \xhookrightarrow{i_R} R$ be a weakly Π-embedded combinatorial rule and $m\colon L \to G$ an embedded match where G is an embedded combinatorial graph. We write $r^{\oplus m}$ for the topological extension of r along the match m, m_Π for the completion of the match, and m'_Π for the comatch

of the double pushout diagram describing the topological extension. To avoid the harder-to-read notations $(L \cap R)^{\oplus m}$, we write $K = L \cap R$ in the proof.

We show that the two conditions of partially embedded rules hold (Definition 74):

- Partially embedded components
- Full orbits of transformed embeddings

▶ *Partially embedded components*

Since r is a weakly Π-embedded combinatorial rule, it satisfies the condition of embedded components (Definition 74). In particular, L is an embedded graph and, thus, a partially embedded graph. The constraint of orbit consistency ensures that any two nodes in the same embedding orbit cannot have distinct embedding values. Besides, since L satisfies the constraint of uniqueness, each node of L is the source of at most π-attribution arc not modified by the topological extension. The topological extension adds nodes to $L^{\oplus m}$ (compared to L) without embedding value. Therefore, all nodes of $L^{\oplus m}$ are the source of at most one π-attribution arc. The condition of non-overlap reduced to the topological extension via the Lemma 13 ensures that any two nodes in the same embedding orbit have the same embedding value. Thus, $L^{\oplus m}$ is a partially embedded graph.

The nodes added to $L^{\oplus m}$ (compared to L) are also added to $K^{\oplus m}$ and $R^{\oplus m}$. The nodes are added without embedding values in both graphs, so the reasoning on L holds. Thereafter, $L^{\oplus m}$, $K^{\oplus m}$, and $R^{\oplus m}$ are partially embedded graphs.

▶ *Full orbits of transformed embeddings*

Consider a node v of K such that $\pi_{L^{\oplus m}}(v) \neq \pi_{R^{\oplus m}}(v)$, Let w be a node in $R^{\oplus m}\langle o_\pi\rangle(v)$. We prove that w is the source of exactly one i-arc in $R^{\oplus m}\langle o_\pi\rangle(v)$ for each i in $orbo_\pi$ by distinguishing on the origin of w. It can either be a node added by the orbit completion, the image of an added node in $R \setminus K$, or the image of a preserved node in K.

Assume w is a node added by the orbit completion. Then, because G is a combinatorial graph and satisfies the incident arcs constraint, the topological extension ensures that w is the source of exactly one i-arc in $R^{\oplus m}$ per i in $\langle o_\pi\rangle$.

Otherwise, the injectivity of the topological part of the comatch $m'_\Pi\colon R \to R^{\oplus m}$ ensure there exist a unique node w_R in R such that $m'_\Pi(w_R) = w$. If w_R is an added node of $R \setminus K$, the incident arcs condition (Definition 34) on r guarantees that w_R is the source of exactly one ar per label in $0..n$. In particular w_R is the source of exactly one i-arc for each i of $\langle o_\pi\rangle$. These arcs yields arcs in $R^{\oplus m}$ via m'_Π, such that w is the source of exactly one i-arc in $R^{\oplus m}$ per i in $\langle o_\pi\rangle$.

Finally, we consider the case where w_R is a preserved node of K and i a label from $\langle o_\pi\rangle$.

- Thanks to the incident arcs condition (Definition 34) on r, if w_R is the source of an i-arc in L then it is also in R and w is the source of an i-arc in $R^{\oplus m}$.
- Otherwise, w is the source of an i-arc in $L^{\oplus m}$, added by the $\langle o_\pi\rangle$-completion. By construction of the topological extension of r along m, this i-arc is also in $K^{\oplus m}$ and $R^{\oplus m}$.

Either way, w is the source of an i-arc in $R^{\oplus m}$. Because G is a combinatorial graph and because $m\colon L \to G$ is an embedded match, w is not the source of another i-arc in $R^{\oplus m}$.

Therefore, $r^{\oplus m}$ satisfies the condition of the full orbits of transformed embeddings.

Thereafter, if $r = \mathrm{L} \hookleftarrow \mathrm{K} \hookrightarrow \mathrm{R}$ is a weakly Π-embedded combinatorial rule and $m\colon \mathrm{L} \to \mathrm{G}$ an embedded match where G is an embedded combinatorial graph, then the topological extension $r^{\oplus m}$ of r along the match m satisfies conditions of the partially embedded components, full orbits of transformed embeddings, and embedding of extended orbits. □

Appendix C

Analysis of the topological folding algorithm

We provide the complete correctness and complexity analysis of the topological folding algorithm. All proofs are taken from [138].

C.1 Correctness analysis

Given a rule in Jerboa, i.e., a rule scheme, its application is formally described as a graph transformation using results from category theory. Among these results, the uniqueness (up to isomorphisms) of the result Gmap is guaranteed. Similarly, the inferred rule's uniqueness from its instantiation is also guaranteed, although the uniqueness needs to be considered up to equivalent relabeling on the provided instantiation. Let us assume that the algorithm provides a rule scheme $r : \mathrm{L} \rightarrowtail \mathrm{R}$ from two instances G and H of an object on a given orbit type $\langle o \rangle$ and a given dart a. It is sufficient to show that the instantiation of L and R on $\mathrm{G}\langle o \rangle(a)$ are respectively isomorphic to G and H.

The provided proof exploits the instantiation process defined with relabeling functions, reusing the various notations and constructions introduced in Chapters 6 and 7. To ease the proof, we also introduce some additional notations.

- P denotes the partial graph scheme during the execution of the algorithm,
- V is the set of nodes of P,
- $\mathrm{V}^{\perp}$ is the subset of nodes from V decorated with $\perp$, i.e., which have not yet seen the construction of their orbit type.
- $\mathrm{V}^{\odot}$ is the subset of nodes from V decorated with a valid orbit type but whose arcs have not yet been extended.
- $\mathrm{V}^{\circledast}$ is the subset of nodes from V decorated with a valid orbit and whose incident arcs have been constructed.

Note that $\mathrm{V}^{\perp}$, $\mathrm{V}^{\odot}$, and $\mathrm{V}^{\circledast}$ forms a partition of V. We can describe the algorithm through its operations on $\mathrm{V}^{\perp}$, $\mathrm{V}^{\odot}$, and $\mathrm{V}^{\circledast}$. Step 1 initializes P as a graph with a single node h decorated with

$\langle o\rangle$. Thus, at the end of this step, $V^{\perp} = \emptyset$, $V^{\odot} = \{h\}$, and $V^{\circledast} = \emptyset$. Step 2.1 moves a node m from $V^{\odot}$ to $V^{\circledast}$ and may add new elements to $V^{\perp}$. Similarly, Step 2.2 moves a node m from $V^{\perp}$ to $V^{\odot}$ without any other modification.

We also use the shorter notation $R_v^{\langle o\rangle}$ to denote the the relabeling function $\langle o\rangle \mapsto \langle o^v\rangle$ for a node v in a graph scheme S.

To prove the correctness of the algorithm, we show two statements, assuming the algorithm runs on a Gmap G with the orbit type $\langle o\rangle$ and initial dart a.

1. If the algorithm provides a graph scheme S, then its instantiation on $\langle o\rangle$ yields the initial graph G.

2. The algorithm fails if there exists no scheme that instantiates in the initial graph G for the type $\langle o\rangle$ with preservation of a.

By induction on the steps of the algorithm, we show that the partial graph P manipulated by the algorithm satisfies the following property $\mathscr{P}(P)$:

1. $\iota^{\langle o\rangle}(P, G\langle o\rangle(a))$ is the unique subgraph of G where (a,h) is mapped onto a.

2. for all nodes v in $V^{\circledast}$, no arc of the form $(b,v) \overset{i}{\bullet\!\!-\!\!\bullet} (c,v')$ exists in G that does not belong to $\iota^{\langle o\rangle}(P, G\langle o\rangle(a))$, where b and c are darts of $G\langle o\rangle(a)$, v' is a node of P, and i a dimension of $0..n$.

Step 1 (Orbit graph and construction of the hook) Step 1 adds a unique node h decorated $\langle o\rangle$ and builds $G\langle o\rangle(a)$ through a graph traversal. Let P_h be the graph that only contains h. By construction $\iota^{\langle o\rangle}(P_h, G\langle o\rangle(a))$ is obtained via the identity function and provides a graph isomorphic to $G\langle o\rangle(a)$. Mapping (a,h) onto a ensures, via the incident ars property on G that $G\langle o\rangle(a)$ is essentially mapped onto itself. Besides, we have $V^{\circledast} = \emptyset$ at the end of the step. Thereafter $\mathscr{P}(P_h)$ holds.

Step 2.1 (Extension of the explicit arcs incident to a node) We consider a partial graph scheme P that satisfies $\mathscr{P}(P)$ and assume that the next action is the construction of the explicit arcs incident to a node m that belongs to $V^{\odot}$. We write $P \circledast m$ for the graph scheme obtained by the addition or extension of arcs for all dimensions not in $\langle o^m\rangle$, assuming the algorithm does not halt on the 'arc failure' case.

Assume that the algorithm does not fail when constructing the explicit arcs incident to the node m. Let i be a dimension of $0..n \setminus \langle o^m\rangle$ and e be the i-arc incident to m added to P. Assume that the addition of e violates the property $\mathscr{P}$. Since all arcs incident to m have not yet been extended, the second condition of $\mathscr{P}$ holds from P. Thus, $\iota^{\langle o\rangle}(P \cup \{e\}, G\langle o\rangle(a))$ is not a subgraph of G when mapping (a,h) onto a. By induction hypothesis, one arc that results from $\iota^{\langle o\rangle}(e, G\langle o\rangle(a))$ does not belong to G. This contradicts the 'instantiation failure' condition from Step 2.1 and $\mathscr{P}(P \cup \{e\})$ holds. Once all the d-links incident to (a,m) have been considered (without failure), m belongs to $V^{\circledast}$. Assume that $\mathscr{P}(P \circledast m)$ does not hold. Since $\mathscr{P}(P \cup \{e\})$ holds for each e arc incident to m, the second condition of $\mathscr{P}(P \circledast m)$ is violated. An arc $(b,m) \overset{i}{\bullet\!\!-\!\!\bullet} (c,m)$ of G is not in the instantiation $\iota^{\langle o\rangle}(P \circledast m, G\langle o\rangle(a))$, where b and c are darts of $G\langle o\rangle(a)$, and i is a dimension. By induction hypothesis, $\mathscr{P}(P)$ held, and i cannot be a dimension of $\langle o^m\rangle$. Otherwise, the construction of $\langle o^m\rangle$ would

have resulted in a failure state. Thus, an i-link is incident to (a,m) in G. Because the algorithm did not stop at the 'arc failure' case, the link is of the form $(a,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (a,m')$ and lead to the creation of an arc $m \bullet\!\!\xrightarrow{i}\!\!\bullet m'$. By the incident arcs property in G, no link $(b,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (b,m')$ can exist. Therefore, the algorithm went into a failure state for 'instantiation failure'. By contradiction, $\mathscr{P}(\mathrm{P} \circledast m)$ holds.

Assume that the algorithm fails to run Step 2.1. In the case of 'arc failure', there is a dimension i in $0..n \setminus \langle o^m \rangle$ such that the i-link incident to (a,m) is of the form $(a,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (b,m')$ with b ($\neq a$) a dart of $\mathrm{G}\langle o \rangle(a)$ and m' ($\neq m$) a node of P. Such a link cannot be obtained by either an explicit or an implicit arc. An implicit arc would instantiate into a link $(a,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (b,m)$, which would not belong to G by the incident arcs constraint. An explicit arc would instantiate into a link $(a,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (a,m')$, which would not belong to G either, by the incident arcs constraint. Both cases would result in $\iota^{\langle o \rangle}(\mathrm{P} \circledast m, \mathrm{G}\langle o \rangle(a))$ not being a subgraph of G, once (a,h) is mapped onto a. In the case of 'instantiation failure', there are a dimension i in $0..n \setminus \langle o^m \rangle$ and a dart c in $\mathrm{G}\langle o \rangle(a)$ such that the i-link incident to (a,m) gave rise to an arc $m \bullet\!\!\xrightarrow{i}\!\!\bullet m'$, but no link $(c,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (c,m')$ exists in G. The instantiation of the arc $m \bullet\!\!\xrightarrow{i}\!\!\bullet m'$ would create an arc $(c,m) \bullet\!\!\xrightarrow{i}\!\!\bullet (c,m')$ in $\iota^{\langle o \rangle}(\mathrm{P} \circledast m, \mathrm{G}\langle o \rangle(a))$. Therefore, $\iota^{\langle o \rangle}(\mathrm{P} \circledast m, \mathrm{G}\langle o \rangle(a))$ would not be a subgraph of G, once (a,h) is mapped onto a.

Step 2.2 (Construction of the orbit type decorating a node) We consider a partial graph scheme P that satisfies $\mathscr{P}(\mathrm{P})$ and assume that the next action is the construction of the orbit type associated with a node m that belongs to $\mathrm{V}^{\perp}$. Let $\mathscr{P} \odot m$ be the graph scheme obtained after the addition of $\langle o^m \rangle$ to m.

Assume that the algorithm does not fail when constructing the orbit type $\langle o^m \rangle$. The addition of $\langle o^m \rangle$ to m moves m from $\mathrm{V}^{\perp}$ to $\mathrm{V}^{\odot}$ without modifying $\mathrm{V}^{\circledast}$. Thus, the second condition of $\mathscr{P}(\mathrm{P} \odot m)$ holds by induction hypothesis on P. Since the algorithm did not fail, for all i in $\langle o \rangle$ such that $\mathrm{R}_m^{\langle o \rangle}(i) = j$ and $j \neq \_$, for all b, c in $\mathrm{G}\langle o \rangle(a)$ such that a link $b \bullet\!\!\xrightarrow{i}\!\!\bullet c$ exists in $\mathrm{G}\langle o \rangle(a)$, a link $(b,m) \bullet\!\!\xrightarrow{j}\!\!\bullet (c,m)$ exists in $\iota^{\langle o \rangle}(\mathrm{P} \odot m, \mathrm{G}\langle o \rangle(a))$. In other words, the extension from P to $\mathrm{P} \odot m$ results, through the instantiation mechanism in the addition of all links of $\mathrm{R}_m^{\langle o \rangle}(\mathrm{G}\langle o \rangle(a))$. Assume that the addition of these links violates $\mathscr{P}(\mathrm{P} \odot m)$. Only the first condition can be violated, meaning that $\iota^{\langle o \rangle}(\mathrm{P} \odot m, \mathrm{G}\langle o \rangle(a))$ is not a subgraph of G, once (a,h) is mapped onto a. By induction hypothesis, $\mathscr{P}(\mathrm{P})$ holds. Therefore, one of the added links does not belong to G. Let i and j be the dimensions such that the superfluous link is of dimension j mapped from a i-link. Since the algorithm did not stop on the 'relabeling failure' case, the relabeling means that all links $b \bullet\!\!\xrightarrow{i}\!\!\bullet c$ in $\mathrm{G}\langle o \rangle(a)$ results in $(b,m) \bullet\!\!\xrightarrow{j}\!\!\bullet (c,m)$ in G, contradicting that the link does not belong to G. Therefore, $\mathscr{P}(\mathrm{P} \odot m)$ holds.

Assume that the algorithm fails to run Step 2.2 when constructing the orbit type $\langle o^m \rangle$. The 'relabeling failure' was triggered by a dimension i from $\langle o \rangle$. Let b_i be the other extremity of the i-link incident to a. Because the algorithm failed a relabeling was decided for i. Let it be j, i.e., $\mathrm{R}_m^{\langle o \rangle}(i) = j$. The failure state also means there are b and c in $\mathrm{G}\langle o \rangle(a)$ such that $b \bullet\!\!\xrightarrow{i}\!\!\bullet c$ belongs to $\mathrm{G}\langle o \rangle(a)$ but $(b,m) \bullet\!\!\xrightarrow{j}\!\!\bullet (c,m)$ does not belong to G. Therefore, $\iota^{\langle o \rangle}(\mathrm{P} \odot m, \mathrm{G}\langle o \rangle(a))$ would not be a subgraph of G, once (a,h) is mapped onto a. □

Termination Since G is finite, the algorithm stops eventually. By induction, when it stops, all nodes of $\mathscr{S}$ belong to $\mathrm{V}^{\circledast}$, i.e., have an orbit type and extended incident arcs. Since $\mathscr{P}(\mathscr{S})$ holds, $\iota^{\langle o \rangle}(\mathscr{S}, \mathrm{G}\langle o \rangle(a))$ is the unique subgraph of G where (a,h) is mapped onto a. Besides, for all nodes v of $\mathscr{S}$, there exists no arc of the form $(b,v) \bullet\!\!\xrightarrow{i}\!\!\bullet (c,v')$ in G that does not belong to $\iota^{\langle o \rangle}(\mathscr{S}, \mathrm{G}\langle o \rangle(a))$,

where b and c are darts of $\mathrm{G}\langle o\rangle(a)$, v' is a node of $\mathscr{S}$, and i a dimension of $0..n$. Since G is connected, $\iota^{\langle o\rangle}(\mathscr{S}, \mathrm{G}\langle o\rangle(a))$ is the complete graph G, once (a,h) is mapped onto a. □

Folding rules The discussion targets the presentation done in Section 7.2. The input of the algorithm is therefore considered to be a Gmap. The proof does not account for the addition of the κ arcs from the joint representation, nor the dart selection when describing a local operation. Both considerations mean that some darts may be missing neighbors for a given dimension. Intuitively, when we consider the copy of the initial dart a within the set of darts associated with the current node, additional verifications need to be realized. First, we need to ensure that the instantiation of either an explicit or an implicit arc does not create additional links in the joint representation. Secondly, we have to check that the instantiation of the graph scheme spans all links of the joint representation, meaning none got forgotten. This verification is achieved by establishing the complete neighborhood of all darts associated with a node. Given the neighborhoods, we first assert that for each dimension (including κ) either all the darts have a neighbor or none have. Then, we work with the restricted set of dimensions where darts indeed have a neighbor. The proof straightforwardly extends with this modification. □

C.2 Complexity analysis

The algorithm runs in time $O(|\mathrm{G}|)$ for a Gmap G. The complexity does not depend on the size of the orbit type $\langle o\rangle$ nor the size of the orbit graph $\mathrm{G}\langle o\rangle(a)$. Step 1 is achieved as a breadth-first search on a in G and takes time $O(|\mathrm{G}\langle o\rangle(a)|)$. Step 2.1 requires checking for each dimension not in $\langle o^m\rangle$ whether all darts in the instantiation of m have a coherent incident arc for that dimension. The number of dimensions is bounded by n, and there are $|\mathrm{G}\langle o\rangle(a)|$ darts for which the incident arcs should be checked. Thus, Step 2.1 requires $O(|\mathrm{G}\langle o\rangle(a)|)$ time. Step 2.2 requires building the relabeling function. When we build a node in the scheme, we store, for each dart, its associated dart in $\mathrm{G}\langle o\rangle(a)$. Therefore, building a relabeling means comparing the dimensions of the arcs incident to a with the arcs incident to its associated dart in the current node. Since there are at most n arcs incident to a, building the relabeling takes time $O(1)$. The consistency of the relabeling must be checked for each dart associated with the current node. By construction, there are $|\mathrm{G}\langle o\rangle(a)|$ such darts. Again, the number of possible arcs to check is bounded by n, resulting in a complexity of $O(|\mathrm{G}\langle o\rangle(a)|)$ for Step 2.2. Step 2 requires iterating Steps 2.1 and 2.2 until the whole graph has been traversed. The number of iterations is directly equal to the number of nodes in the resulting rule scheme. Each node in the rule scheme corresponds to $|\mathrm{G}\langle o\rangle(a)|$ darts in the initial Gmap. Therefore, the folded representation will have $\frac{|\mathrm{G}|}{|\mathrm{G}\langle o\rangle(a)|}$ nodes and the overall complexity is $O(|\mathrm{G}|)$. On two instances that compose a rule, the complexity is the same for both the left-hand side and the right-hand side, unified using the κ-arcs. Thus, the full complexity to build the rule scheme from a before Gmap G and an after Gmap H is $O(|\mathrm{G}| + |\mathrm{H}|)$, i.e., a linear complexity. □

Appendix D

Inferred embedding expressions

We provide the missing inferred embedding expressions of Chapter 8.

D.1 Inferred embedding expressions for the (2,2,2)-Menger sponge

```
// n1
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.20000000000000062);
res.add(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.39999999999999947);
res.add(p1);
Point3 p3 = Point3::middle(<0, 1, 2>_position(n0));
p3.scale(0.4);
res.add(p3);
return res;
```

```
// n2
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.20000000000000018);
res.add(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.7999999999999998);
res.add(p1);
return res;
```

```
// n4
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.19999999999999984);
res.add(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.40000000000000013);
res.add(p1);
Point3 p2 = Point3::middle(<0, 1>_position(n0));
p2.scale(0.4);
```

```
res.add(p2);
return res;
```

```
// n5
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.2000000000000003);
res.add(p0);
Point3 p2 = Point3::middle(<0, 1>_position(n0));
p2.scale(0.4);
res.add(p2);
Point3 p3 = Point3::middle(<0, 1, 2>_position(n0));
p3.scale(0.4);
res.add(p3);
return res;
```

```
// n9
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.5999999999999998);
res.add(p0);
Point3 p2 = Point3::middle(<0, 1>_position(n0));
p2.scale(0.4);
res.add(p2);
return res;
```

```
// n11
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.5999999999999996);
res.add(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.40000000000000036);
res.add(p1);
return res;
```

```
// n17
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.6000000000000005);
res.add(p0);
Point3 p3 = Point3::middle(<0, 1, 2>_position(n0));
p3.scale(0.4);
res.add(p3);
return res;
```

```
// n18
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.19999999999999996);
res.add(p0);
Point3 p2 = Point3::middle(<0, 1>_position(n0));
```

```
p2.scale(0.8);
res.add(p2);
return res;
```

```
// n22
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.2000000000000003);
res.add(p0);
Point3 p3 = Point3::middle(<0, 1, 2>_position(n0));
p3.scale(0.8);
res.add(p3);
return res;
```

Appendix E

Synthèse en français

Dans cette thèse, nous nous sommes intéressés à des mécanismes d'assistance à la conception et à la spécification d'opérations de modélisation géométrique. Nous avons notamment étudié leur formalisation comme des règles de transformation de graphes.

Dans une première partie, nous avons étudié la conception d'un langage dédié utilisant des règles de transformation de graphes, le but étant d'allier une apparente simplicité d'expression d'une opération avec la rigueur sous-jacente des transformations de graphes. Nous avons étudié une représentation par graphe des cartes généralisées et orientées, usuellement définies à l'aide de permutations. Chaque permutation décrivant une relation de voisinage pour une dimension donnée, les graphes obtenus sont étiquetés sur les arcs par les dimensions correspondantes. La représentation par graphe suppose le traitement de contraintes de cohérence topologique pour lesquelles nous avons étudié des conditions sur les règles pour garantir la présentation des contraintes du modèle. Il s'agit en effet d'assurer qu'une opération appliquée sur un objet bien formé produit un objet bien formé. En particulier, nous avons réduit les conditions suffisantes existantes à des conditions nécessaires et suffisantes. Nous avons ainsi pu obtenir une généralisation du traitement algébrique de la réécriture des cartes généralisées vers un modèle comprenant aussi les cartes orientées, formalisation directe de la structure de demi-arêtes prépondérante en modélisation géométrique. Outre la préservation de la cohérence du modèle, l'étude des règles de réécriture de graphes comme support de formalisation des opérations de modélisation géométrique nous a conduit à étendre le cadre de la réécriture par doubles sommes amalgamées afin d'obtenir un niveau de généricité suffisant vis-à-vis des attentes de la communauté de modélisation géométrique. Cette généricité a été obtenue via des schémas de règles qui permettent d'abstraire la topologie sous-jacente de l'objet modifié. Nous avons présenté une extension semi-globale de la réécriture usuelle par DPO en incorporant un produit de graphes simulant l'application d'une fonction de renommage. Nous avons ainsi obtenu des règles de réécriture en adéquation avec les opérations usuelles manipulées en modélisation géométrique, mais complètement formalisées par le biais des catégories. À cette description topologique des objets s'ajoutent des informations géométriques que l'on représente à l'aide d'attributs sur les graphes pour encoder les fonctions de plongements usuelles décrites sur les cellules topologiques. En effet, la représentation des objets et leur manipulation concrète dans un domaine applicatif requièrent la manipulation des valeurs associées à l'objet, a minima des positions géométriques pour les sommets de l'objet. Les cellules topologiques d'une carte généralisée sont des cas particuliers d'orbites (au sens des orbites d'une permutation). Dans l'approche par graphes, ces orbites correspondent

alors à des sous-graphes induits par un ensemble de dimensions. L'ensemble des dimensions constitue le type de l'orbite. L'ajout de plongement à une carte généralisée suppose que les valeurs de plongements soient identiques au sein d'une orbite du type correspondant. Une contrainte de cohérence géométrique est donc adjointe au modèle topologique. Nous avons établi des conditions de préservation pour ces contraintes que nous avons dans un deuxième temps relaxées via un mécanisme de complétion des règles reposant sur une extension topologique du motif filtré et une propagation géométrique des termes. In fine, nous avons établi un mécanisme de préservation de la cohérence topologique et géométrique par le biais de conditions syntaxiques vérifiées statiquement sur les règles.

Dans une seconde partie, nous avons présenté un mécanisme d'inférence d'opérations. Avec l'intuition qu'une opération peut être facilement décrite via un croquis ou un exemple, nous avons proposé une méthode de reconstruction d'une opération à partir d'un objet initial et d'un objet cible. Notre mécanisme d'inférence exploite la régularité des cartes généralisées et du langage dédié. Puisque la topologie et la géométrie possèdent une représentation distincte aussi bien dans la structure de manipulation des objets (Gcartes, Ocartes) que dans le langage de description des opérations de modélisation, nous avons étudié séparément l'inférence des modifications topologiques et géométriques liées à une opération. Plus précisément, nous avons envisagé la question de l'inférence d'opérations topologiques comme la construction inverse de la spécialisation d'un schéma de règle vers une opération. Il faut ainsi penser l'inférence d'une opération comme la construction d'un quotient de graphe guidé par le motif topologique fourni comme entrée de l'algorithme. L'algorithme repose sur un simple parcours de graphe. Nous avons ainsi obtenu une solution univoque pour la reconstruction de la topologie. Une opération topologique modifie uniquement la structure du graphe manipulé tandis qu'une opération géométrique implique la modification de valeurs associées aux cellules topologiques. Si ces valeurs sont numériques, l'ensemble des modifications possibles devient alors inexplorable. Plus généralement, étant donné le type de géométrie et la nature des modifications appliquées, les solutions sont multiples. Nous avons traité le cas des transformations affines de valeurs évoluant dans un espace vectoriel que nous avons résolu comme un problème de satisfaction de contraintes. Nous avons implanté les mécanismes d'inférence topologique et géométrique dans la plateforme Jerboa qui permet la conception de modeleurs. Ainsi, bien que les constructions dans Jerboa puissent sembler ésotériques, et malgré la supposition qu'un langage à base de règle puisse rendre plus accessible la description des opérations de modélisation, l'implémentation d'un module d'inférence d'opérations dans Jerboa a aussi vocation à simplifier la conception de nouvelles opérations. La première partie de cette thèse a ainsi permis de construire un cadre formel qui est de facto masqué à l'utilisateur, mais demeure nécessaire pour la conception d'opérations de modélisations géométriques via notre mécanisme d'inférence.