Planning Abstract Consistency Repairs Between Heterogeneous Models

Henriette Färber[1], Romain Pascual[2], Shobhit Singh[1(✉)], Terru Stübinger[1], and Mattias Ulbrich[1]

[1] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[2] MICS, CentraleSupélec, Université Paris-Saclay, Saclay, France
Corresponding author:(✉) shobhit.singh@kit.edu

Abstract. Asserting, maintaining, and repairing consistency between heterogeneous models is a central challenge in model-driven development. Consistency is often defined at the semantic level, but consistency repair techniques typically operate at the syntactic level via language-specific model transformations. This mismatch motivates a shift of perspective: consistency repair should be planned at the semantic level. Building on the notion of observables as a shared semantic space, we introduce contracts as abstractions of model transformations, and we prove key correctness properties. Contracts then capture effects on observable states instead of syntactic changes. We obtain a uniform formalism in which consistency repair can be formulated as a planning problem over set-valued observable states under the goal of restoring consistency. We provide a concrete representation of observable states using abstract domains and cover a representation of contracts via pre- and post-conditions expressed in a logic suitable for automated analysis. This representation allows the use of off-the-shelf formal-methods techniques, such as SAT/SMT-based bounded search and symbolic model checking, to synthesize repair plans. We demonstrate the feasibility of the approach by implementing a prototype in nuXmv, showing how repair plans can be automatically synthesized for a representative heterogeneous modeling scenario.

Keywords: Model-Driven Development, Consistency Repair, Contract-Based Planning, Observable Semantics, Cyber-Physical Systems

1 Introduction

The design of modern cyber-physical systems requires heterogeneous models from very different domains, each capturing a different view of the system, such as requirements, software architecture, dynamic behavior, physical properties, or timing constraints. In Model Driven Development (MDD), these views are typically maintained by different teams, modeled in different languages with different tools, and evolved independently and concurrently [25]. While necessary in practice, this separation introduces a central challenge: models may overlap in

their meaning and must be kept consistent. As soon as one or more models are evolved, the remaining heterogeneous model infrastructure needs to be repaired to restore consistency.

While notions of consistency are often most naturally defined at the semantic level [34] (e.g. "the maximum weight of the car must not exceed the weight admissible for the brake setup"), repair techniques usually operate at the syntactic level, for example by executing model transformations on individual artifacts. This creates a fundamental mismatch. Model transformations are typically language-specific, operational, and tightly coupled to concrete model representations, whereas consistency conditions may refer to semantic properties shared across several modeling domains. As a consequence, achieving consistency repairs across heterogeneous artifacts becomes difficult: one must reason simultaneously about different modeling languages, different transformation mechanisms, and the combined effect of local edits on global consistency.

This mismatch is not only conceptual but also methodological: it raises the question of what properties a suitable repair approach should satisfy in heterogeneous development settings. Following the classification of Macedo et al. [32], such approaches must support multi-model domains with inter-model constraints, remain metamodel-independent, allow user-defined constraints, and handle consistency relations beyond purely syntactic correspondences.

At the same time, these requirements must be met in a setting where consistency is defined at the semantic level, while repair actions are traditionally operational and model-specific. This tension motivates the need for a representation of repair that operates directly at the semantic level while remaining amenable to automated analysis. The approach proposed in this paper addresses these requirements by operating on a shared semantic representation while remaining independent of concrete modeling languages and transformation mechanisms.

Moreover, consistency repair is not merely the problem of finding a consistent semantic state. In practice, engineers need to understand how a system evolves into a repaired one and which changes are to be applied. The primary artifact of consistency management is therefore not a repaired state but a repair plan: a traceble sequence of admissible changes leading to consistency.

These observations lead to a shift of perspective: if consistency is defined semantically, then consistency repair should also be planned semantically. Rather than searching directly in the space of concrete model edits, we search in a shared semantic space. This shift constitutes the central idea of the paper. In recent work [15], *observables* have been proposed as a semantic interface for this purpose. Instead of comparing heterogeneous models syntactically, one projects shared system-level properties from the individual models and defines consistency in terms of the compatibility of the associated observable values. Observables then yield a uniform, domain-agnostic notion of consistency and provide a shared, interdisciplinary cognitive space for domain experts [22]. Recent work has also proposed an ontology-guided retrieval-augmented generation pipeline to automatically extract observables from heterogeneous engineering artifacts [30].

We extend this framework with *contracts* as the key abstraction for repairing consistency at the level of observable semantics. In our setting, a contract is not a program-level behavioral specification of a routine [33], but an abstract summary of a class of model transformations. It describes how the observable state of a system may change, without committing to how that change is realized in any particular modeling language or transformation formalism. In this sense, contracts capture *what* a repair step achieves semantically, rather than *how* it is operationally implemented, and can be understood as abstractions of families of concrete model transformations that have the same observable effect. Contracts can then be composed into repair plans that progressively restore consistency, while allowing temporary inconsistencies along the way.

This contract-based view enables a clean separation of concerns. Observables define a semantic abstraction layer over heterogeneous models. Consistency is expressed as an explicit relation over observable states. Contracts provide an operational approximation of admissible transformations at the semantic level. Finally, repair planning amounts to searching for sequences of contract applications that transform an inconsistent observable state into a consistent one. This separation allows the planning process to operate independently of concrete modeling languages and transformation mechanisms. The link to model-level behavior is obtained through soundness requirements.

Beyond separation of concerns, this contract-based approach offers several advantages. First, it provides a uniform and domain-independent representation of repair actions. Second, it separates repair planning from repair execution: the planner reasons over abstract contracts and observable states, while the refinement to concrete model transformations can be addressed independently. Third, it supports modularity: operations can be restricted to a single model (or a few models). This separation is especially useful in distributed development settings, where teams may wish to expose the effect of permissible changes without coupling internal editing procedures.

This paper lays the formal foundations for repairing inter-model consistency as a planning problem based on the framework of observables [15]. States correspond to valuations of observable variables, actions correspond to contracts, and goal states correspond to satisfaction of the consistency relation. Then, consistency repair boils down to finding a sequence of contract applications that leads from an inconsistent state to a consistent one. To make this amenable to automated analysis, we encode observable states using abstract domains and represent contracts as transition relations over these domains. The framework naturally accommodates abstraction, nondeterminism, and temporary inconsistencies during repair. Moreover, once cast in this form, repair planning can be tackled using standard formal-methods technology, such as SAT/SMT-based bounded search and symbolic model checking [8, 24, 29, 38].

The main contribution of this paper is a contract-based formalization of consistency repair planning for heterogeneous models. We make the following contributions:

- the formal introduction of contracts as semantic abstractions of model transformations over observable states (Sect. 4);
- the definition of key correctness properties of contracts, including admissibility, locality, well-formedness, and soundness, and study their behavior under sequential composition (Sect. 4);
- the formulation of consistency repair as a planning problem over contract-labeled transitions in the shared space of observables (Sect. 4);
- the encoding of this planning problem as a reachability problem in a finite-state transition system, enabling the use of off-the-shelf solver technology (Sect. 5);
- the illustration of the approach on a prototype encoding using nuXmv, showing how repair plans can be synthesized automatically (Sect. 6).

2 Running Example

Throughout the paper we consider a collection of heterogeneous models of parts of an electric vehicle. It consists of three engineering artifacts describing complementary aspects of the same system: a parametric CAD model of the vehicle body, an ECAD schematic including nominal information about the batteries and topology of the wiring and a suspension spreadsheet specifying admissible load constraints. Each artifact captures different engineering knowledge while sharing information with the others through physical quantities such as voltage and weight. For simplicity, we only consider the weight of the various components such that consistency requirement is that the total vehicle weight does not exceed the maximum weight supported by the suspension.

Each engineering artifact admits its own editing operations. For example, an engineer may select a different battery cell, modify the pack topology, resize the vehicle body, or choose a different material. By contrast, the suspension models is treated as fixed specifications. The goal throughout the paper is to repair inconsistencies between these heterogeneous artifacts.

3 Preliminaries

Before introducing contract-based consistency repair in Sect. 4, we recall the formal setting on which our approach is built: the set-theoretic formalization of consistency [34], and observables as semantic abstractions of system properties [15]. We refer the reader to the original sources for a more comprehensive discussion.

Notation 1 *Given a set I and an I-indexed family $\mathcal{X} = (X_i)_{i\in I}$ of sets, we write $\prod_I X_i$ (or $\prod X$ for short) for the Cartesian product of the sets in $\mathcal{X}$. The elements of $\prod_I X_i$ are tuples $(x_i)_{i\in I}$, where for every $i \in I$, $x_i \in X_i$. In case an index set I has been established, we simplify the notation for the sake of readability and write X_I and x_I instead of $(X_i)_{i\in I}$ and $(x_i)_{i\in I}$ respectively. For $i \in I$, the i-th coordinate of x_I is accessed as x_i.*

In the paper, we consider a multi-sorted signature $\Sigma = (S_\Sigma, F_\Sigma)$ with a fixed interpretation $\mathcal{A}$, i.e., a fixed Σ-algebra. We write $[\![s]\!]_\mathcal{A}$ for the carrier set associated with a sort $s \in S_\Sigma$ and $[\![\sigma]\!]_\mathcal{A} \colon \prod_{1 \le i \le k} [\![s_i]\!]_\mathcal{A} \to [\![s]\!]_\mathcal{A}$ for the function associated with the function symbol $\sigma \colon \prod_{1 \le i \le k} s_i \to s$ in F_Σ.

3.1 Heterogeneous Model Semantics

The VITRUVIUS approach introduces the virtual single underlying model (V-SUM) [25], which represents a system as an internal collection of models that externally behave as a single model, thanks to consistency-preservation mechanisms. Formally, we follow the set-theoretic foundations of [34] and define a V-SUM Meta-Model (V-SUMM) as a pair $\mathcal{M} = (\Pi M, CR)$, where ΠM is the Cartesian product of the individual internal meta-models. Each meta-model is taken to be the set of possible models of its domain, so that the consistency relation $CR \subseteq \Pi M$ can be a simple relation specifying which tuples of models are considered consistent.

Example 1 (Models of a Simplified Electric Vehicle). We consider a simplified electric vehicle described by the following V-SUM

$$m = (m_{\text{body}}, m_{\text{propulsion}}, m_{\text{suspension}}) \in M_{\text{body}} \times M_{\text{propulsion}} \times M_{\text{suspension}},$$

where m_{body} is the CAD model of the body, $m_{\text{propulsion}}$ is the propulsion ECAD schematic, and m_{sus} is the suspension spreadsheet.

The nature of a consistency relation can be quite diverse: it may be a purely syntactical condition, such as requiring that all names referred to in one model are declared in another model. To reason about consistency semantically, Pascual et al. [34] introduced *abstract semantics* as mappings $[\![\cdot]\!] \colon M \to \mathbb{S}$ from a meta-model M to a semantic domain $\mathbb{S}$.

3.2 Observables

Observables provide a way to extract semantic information from models while keeping them otherwise abstract. Rather than assigning a single value to a model, an observable captures the set of values compatible with the model's information. We assume a set O of predefined observables about the system and, for each observable $o \in O$, a non-empty set of possible values V_o.

Definition 1 (Observable Semantics – [15, Def 8]). *Let M a meta-model. An observable $o \in O$ yields a function $o : M \to \mathscr{P}(V_o)$, where $\mathscr{P}(V_o)$ is the powerset of V_o, called an* observable semantics *on M. V_o is then called the* value space *of o.*

For a family $(M_i)_{i \in I}$ of meta-models, an observable o induces a family $(o_i \colon M_i \to \mathscr{P}(V_o))_{i \in I}$ of observable semantics.

Here, $o_i(m_i)$ denotes the set of admissible values of the observable o according to the model m_i.

Example 2 (Observable Semantics). Consider the observable w^{total} describing the total vehicle weight, with value space $V_{w^{\text{total}}} = \mathbb{R}_+$. Different models constrain this observable differently: the body and propulsion models impose lower bounds, while the suspension model imposes both a lower bound on its own weight and an upper bound on the supported load.

For a specific vsum $m = (m_{\text{body}}, m_{\text{propulsion}}, m_{\text{suspension}})$ these semantics could evaluate to the following real-valued intervals

$$\begin{aligned} w^{\text{total}}_{\text{body}}(m_{\text{body}}) &= [80, \infty), \\ w^{\text{total}}_{\text{propulsion}}(m_{\text{propulsion}}) &= [250, \infty), \\ w^{\text{total}}_{\text{suspension}}(m_{\text{suspension}}) &= [50, 500]. \end{aligned}$$

As a special case of abstract semantics, observable semantics induce a consistency relation requiring that models agree on at least one admissible value for the semantics.

Definition 2 (Observable-induced Consistency – [15, Def 9]). *Any observable o induces a consistency relation $CR_o \coloneqq \{m_I \in \Pi M \mid \bigcap_I o_i(m_i) \neq \emptyset\}$.*

Example 3 (Observable-induced Consistency). For VSUM from Ex. 2,

$$\begin{aligned} & w^{\text{total}}_{\text{body}}(m_{\text{body}}) \cap w^{\text{total}}_{\text{propulsion}}(m_{\text{propulsion}}) \cap w^{\text{total}}_{\text{suspension}}(m_{\text{suspension}}) \\ &= [250, 500] \\ &\neq \emptyset \end{aligned}$$

and the models are therefore consistent with respect to w^{total}.

In many cases, system properties are derived from others, which is captured by *compound observables*, defined using an algebraic signature [15]. By contrast, the observables presented so far are called *simple*.

To build compound observables, functions from the underlying algebra are lifted to observables, meaning that we apply the associated operations of $\mathcal{A}$ pointwise. For this, we assume that each observable o is associated with a fixed sort $s \in S_\Sigma$ such that its value space is $[\![s]\!]_{\mathcal{A}}$.

Definition 3 (Pointwise Lifted Interpretation – [15, Def 12]). *Let $\sigma : \prod_K s_k \to s$ be a function symbol in F_Σ and $(o^k)_{k \in K}$ be observables with value sets $([\![s_k]\!]_{\mathcal{A}})_{k \in K}$. Then for each $i \in I$, the* pointwise lifted interpretation $\sigma^\bullet[o^K]_i$ *of σ on $(o^k)_{k \in K}$ for M_i is defined as*

$$\sigma^\bullet[o^K]_i(m_i) \coloneqq \left\{ [\![\sigma]\!]_{\mathcal{A}}(v_K) \mid v_K \in \prod_K o^k_i(m_i) \right\} .$$

The pointwise lifting computes a new observable semantics from its inputs. *Compound observables* further allow combining multiple computations over different subsets of observables.

Definition 4 (Compound Observable – adapted from [15, Def 13]). *Let* $(o^k)_{k\in K}$ *be observables with value sets* $(\llbracket s_k \rrbracket_{\mathcal{A}})_{k\in K}$. *A finite set* $\mathcal{C}$ *of index sets* $c \subseteq K$ *together with a function symbol* $\sigma_c : \prod_{\ell\in c} s_\ell \to s$ *in* F_Σ *for each* $c \in \mathcal{C}$ *defines the* compound observable $\sigma^\bullet_{\mathcal{C}}[o^K]$ *with observable semantics* $\sigma^\bullet_{\mathcal{C}}[o^K]_i$: $M_i \to \mathscr{P}(\llbracket s \rrbracket_{\mathcal{A}})$ *given by*

$$\sigma^\bullet_{\mathcal{C}}[o^K]_i(m_i) \coloneqq \bigcap_{c\in\mathcal{C}} \sigma^\bullet_c[o^c]_i(m_i) \ . \tag{1}$$

Example 4 (Compound Observables). Instead of defining w^{total} (Ex. 2) as a simple observable, we can derive it from component-level observables, capturing each individual weight: $o^{\text{body-weight}}$, $o^{\text{motion-weight}}$, and $o^{\text{suspension-weight}}$

$$\begin{aligned} o^{\text{body-weight}}_{\text{body}}(m_{\text{body}}) &= \{w_{\text{body}}\}\,, \\ o^{\text{motion-weight}}_{\text{propulsion}}(m_{\text{propulsion}}) &= \{w_{\text{propulsion}}\}\,, \\ o^{\text{suspension-weight}}_{\text{suspension}}(m_{\text{suspension}}) &= \{w_{\text{suspension}}\}\,. \end{aligned}$$

where $w_{\text{body}}, w_{\text{propulsion}}, w_{\text{suspension}} \in \mathbb{R}_+$ denote concrete values extracted from the respective model. The observable for the weight of a different component remains unconstrained, i.e., $o^{x\text{-weight}}_y(m_y) = \mathbb{R}_+$ if $x \neq y$. We also consider the supported-load observable $o^{\text{supp-load}}$ constrained by $m_{\text{suspension}}$ as

$$o^{\text{supp-load}}_{\text{suspension}}(m_{\text{suspension}}) = [0, w_{\max}],$$

with unconstrained values on the other models.

We assume $add_3 : \mathbb{R}^3_+ \to \mathbb{R}_+ \in F_\Sigma$ and $id : \mathbb{R}_+ \to \mathbb{R}_+$ are in F_Σ, with $\llbracket add_3 \rrbracket_{\mathcal{A}}(a,b,c) = a+b+c$ and $\llbracket id \rrbracket_{\mathcal{A}}(x) = x$. We also consider the family o^K

$$(o^{\text{body-weight}},\ o^{\text{motion-weight}},\ o^{\text{suspension-weight}},\ o^{\text{supp-load}})$$

and the collection of index sets

$$\mathcal{C} = \{\text{weights},\ \{\text{supp-load}\}\}$$

with weights $= \{\text{body-weight},\ \text{motion-weight},\ \text{suspension-weight}\}$. We can then redefine w^{total} as the compound observable $\sigma^\bullet_{\mathcal{C}}[o^K]$ with $\sigma_{\text{weights}} = add_3$ to compute total weight from individual weights and $\sigma_{\{\text{supp-load}\}} = id$ for the range. Its semantics on $M_{\text{suspension}}$ is derived as follows

$$\begin{aligned} &\sigma^\bullet_{\text{weights}}[o^{\text{weights}}]_{\text{suspension}}(m_{\text{suspension}}) \\ &= \{a+b+c \mid (a,b,c) \in \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_{\text{suspension}}\}\} \\ &= [w_{\text{suspension}}, \infty) \end{aligned}$$

$$\begin{aligned} &\sigma^\bullet_{\{\text{supp-load}\}}[o^{\text{supp-load}}]_{\text{suspension}}(m_{\text{suspension}}) \\ &= \{id(x) \mid x \in [0, w_{\max}]\} \\ &= [0, w_{\max}] \end{aligned}$$

and their intersection yields $\sigma^{\bullet}_{\mathcal{C}}[o^K]_{\text{suspension}}(m_{\text{suspension}}) = [w_{\text{suspension}}, w_{\max}]$, which is non empty iff $w_{\text{suspension}}$ does not exceed $w_{\max}$. Carrying the computation on other models, we retrive the observable semantics from Ex. 2, with 80 abstracted as w_{body}, 250 as $w_{\text{propulsion}}$, 50 as $w_{\text{suspension}}$, and 500 as $w_{\max}$.

In Ex. 4, we recasted the total weight observable from Ex. 2 as a compound observable. At a first glance, the computation of the compound observable semantics is an information loss compared to the simpler observables, while not allowing to recover the total weight $w^{\text{total}}(m)$ as $w_{\text{body}} + w_{\text{propulsion}} + w_{\text{suspension}}$. In [15], extensions of Def. 4 allowed this computation either via tuple encoding or meta-model spanning observable semantics. Because our goal is to repair consistency by acting on the observables via abstractions of model edits, we choose instead to keep partially redundant information, at the price of some bookkeeping via the admissibility property (see Defs. 7 and 10).

Compound observables allow the construction of new observables from existing ones while preserving the set-based semantics. In the following, we fix a family of observables $O = (o^k)_{k \in K}$, containing both simple and compound observables.

4 Contract-Based Repair Planning

The standard approach to model editing is to use model transformations that operationalize edits to the model's underlying structure. In particular, model transformations are often domain-specific or tailored to the language used to encode the models, making them difficult to orchestrate across heterogeneous artifacts. However, for consistency repair, we are not interested in the full operational details, but only in their effects, e.g., on the observable semantics. In other words, when consistency itself is defined semantically, we should adopt a semantic perspective on consistency repair and abstract model transformations, focusing on their effects on observable semantics. This abstraction is further justified by the fact that several model transformations may induce the same effect on observables, but only this effect is needed to assert consistency. Thus, we introduce *contracts* as an abstraction of model transformations that capture such effects. Since the observable semantics associate sets of possible values with each model, contracts operate on those sets, meaning the execution operates on unknown concrete values.

4.1 Observable States

Observable semantics (see Def. 1) associate a set of admissible values to each model for a system property. We retain this set-based representation for operational reasoning and introduce *observable states* as configurations of candidate realizations of the system.

Definition 5 (Observable State). *Let $O = (o^k)_{k \in K}$ be a family of simple observables with value spaces $(V_k)_{k \in K}$, and $(M_i)_{i \in I}$ a family of meta-models.*

An observable state *is a family of subsets* $(S_i \subseteq \prod_K V_k)_{i\in I}$.

Further, let $m \in \Pi M$ *be a V-SUM. For each component model* $m_i \in M_i$, *define* $\nu(m_i) = \left\{v \in \prod_K V_k \mid \forall k \in K,\ v_k \in o_i^k(m_i)\right\}$. *Then, the observable state associated with* m *is* $\nu(m) = (\nu(m_i))_{i\in I}$.

Note, this means that the observable state space is $\prod_I \mathscr{P}\left(\prod_K V_k\right)$. We consider observable states as a family of subsets $(S_i)_{i\in I}$, meaning that an observable state can be decomposed per meta-model but not per observable. Indeed, a state naturally accounts for possible correlations between observables within a meta-model, since relational constraints are inherently non-product sets, e.g., that the total weight is smaller than the maximal load or that voltage depends on the internal configuration of the battery cells. However, the construction of the observable semantics externalizes the relation between the values from models of different meta-models via the induced consistency relation, justifying the choice to have decomposability meta-model-wise. The choice also sanctions a locality property on contracts (see Sect. 4.2).

Example 5 (Observable States). If we only consider the three models of Ex. 1 and the observable w^{total} of Ex. 2, then the associated observable state contains the possible weight values, i.e., $(\nu(m_{\text{body}}), \nu(m_{\text{propulsion}}), \nu(m_{\text{suspension}}))$ with $\nu(m_{\text{body}}) = [80, \infty)$, $\nu(m_{\text{propulsion}}) = [250, \infty)$, and $\nu(m_{\text{suspension}}) = [50, 500]$. If we instead consider $o^K = (o^{\text{body-weight}}, o^{\text{motion-weight}}, o^{\text{suspension-weight}})$ with the same models and the abstracted values from Ex. 4, we obtain

$$\begin{aligned}\nu(m_{\text{body}}) &= \{w_{\text{body}}\} \times \mathbb{R}_+ \times \mathbb{R}_+\\ \nu(m_{\text{propulsion}}) &= \mathbb{R}_+ \times \{w_{\text{propulsion}}\} \times \mathbb{R}_+\\ \nu(m_{\text{suspension}}) &= \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_{\text{suspension}}\}\,.\end{aligned}$$

The complete setting simply append the sets from w^{total} to each $\nu(m_i)$.

When accounting for compound observables, any valuation $v_K \in \prod_K V_k$ might not be admissible in the sense that it does not account for the constraint introduced by the construction of compound observables.

Definition 6 (Induced Value Set for Compound Observable). *Let* $(o^k)_{k\in K}$ *be observables with value sets* $(\llbracket s_k \rrbracket_{\mathcal{A}})_{k\in K}$. *Let* $\mathcal{C}$ *be a finite set of index sets* $c \subseteq K$ *together with a function symbol* $\sigma_c : \prod_{\ell\in c} s_\ell \to s$ *in* F_Σ *for each* $c \in \mathcal{C}$. *Let* $\sigma^\bullet_{\mathcal{C}}[o^K]$ *be the associated compound observable (see Def. 4).*

Given an observable state $S = (S_i)_{i\in I}$, *the* induced value set *for* $\sigma^\bullet_{\mathcal{C}}[o^K]$ *on* S *is the set*

$$\llbracket \sigma^\bullet_{\mathcal{C}}[o^K] \rrbracket(S) = \bigcap_{c\in\mathcal{C}} \{\llbracket \sigma_c \rrbracket_{\mathcal{A}}(v_c) \mid v_c \in \pi_c(\cap_{i\in I} S_i)\}$$

where $\pi_c \colon \prod_{k\in K} \llbracket s_k \rrbracket_{\mathcal{A}} \to \prod_{\ell\in c} \llbracket s_\ell \rrbracket_{\mathcal{A}}$ *is the projection on* c.

In the previous definition, S represents the possible valuations of the base observables, e.g., a component of an observable state. Note that compound observables may be defined over already defined compound observables (essentially in an inductive way), which naturally extends to the previous definition.

From now on, we extend the definition of observable states (Def. 5) to allow compound observables, under the hypothesis that compound observables are built over other observables in O. We then restrict the study to admissible observable states, in which the value sets of compound observables are induced value sets.

Definition 7 (Admissible Observable State). *Let $O = (o^k)_{k\in K}$ be a family of observables with value spaces $(V_k)_{k\in K}$.*

Assume each compound observable $\sigma^\bullet_{\mathcal{C}}[o^K] \in O$ is associated with a dependency set $K_\sigma \subseteq K$, namely the set of observables appearing in $\mathcal{C}$, and an output index $k_\sigma \in K$.

An observable state $S = (S_i)_{i\in I}$ is admissible *if for every component $i \in I$ and every compound observable $\sigma^\bullet_{\mathcal{C}}[o^{K_\sigma}] \in O$, the following holds:*

$$[\![\sigma^\bullet_{\mathcal{C}}[o^{K_\sigma}]]\!](\pi_{K_\sigma}(S)) = \bigcap_{i\in I} \{v_{k_\sigma} \mid v \in S_i\}$$

where v_{k_σ} is the component of v at the index k_σ and $\pi_{K_\sigma}(S)$ is the projection on the indices used to define $\sigma^\bullet_{\mathcal{C}}[o^{K_\sigma}]$.

Unravelling the various construction, one observes that $\nu(m) = (\nu(m_i))_{i\in I}$ is generally inadmissible in the presence of compound observables. Indeed, compound observables are computed locally for each model when constructing $\nu(m)$, whereas admissibility reconstructs their value sets from the information shared across all component states (see Ex. 4). To account for this, we introduce the *admissibility closure*. Given an observable state S, let $\overline{S}$ denote the observable state obtained by replacing every compound observable by its induced value set. In the remainder of the paper, we implicitly identify an observable state with its admissibility closure and write $\nu(m)$ in place of $\overline{\nu(m)}$.

Example 6 (Admissibility in the EV system). Building on Ex. 5 with the observable ordered as $o^{\text{body-weight}}$, $o^{\text{motion-weight}}$, $o^{\text{suspension-weight}}$, $o^{\text{supp-load}}$, and w^{total} (and assuming $w_{\text{body}} + w_{\text{suspension}} + w_{\text{propulsion}} \leq w_{\max}$), the state given by

$$\begin{aligned}
\nu(m_{\text{body}}) &= \{w_{\text{body}}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times [w_{\text{body}}, \infty) \\
\nu(m_{\text{propulsion}}) &= \mathbb{R}_+ \times \{w_{\text{propulsion}}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \times [w_{\text{propulsion}}, \infty) \\
\nu(m_{\text{suspension}}) &= \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_{\text{suspension}}\} \times [0, w_{\max}] \times [w_{\text{suspension}}, w_{\max}]
\end{aligned}$$

is inadmissible, but replacing the sets from w^{total} by their induced value set, the state given by

$$\begin{aligned}
\overline{\nu(m_{\text{body}})} &= \{w_{\text{body}}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_+\} \\
\overline{\nu(m_{\text{propulsion}})} &= \mathbb{R}_+ \times \{w_{\text{propulsion}}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_+\} \\
\overline{\nu(m_{\text{suspension}})} &= \mathbb{R}_+ \times \mathbb{R}_+ \times \{w_{\text{suspension}}\} \times [0, w_{\max}] \times \{w_+\}
\end{aligned}$$

is admissible, with $w_+ = w_{\text{body}} + w_{\text{suspension}} + w_{\text{propulsion}}$.

Admissibility is essentially the semantic closure under the compound observable applied to the observable states. It provides some internal coherence of the semantic encoding. From there, we can consider the consistency of an observable state:

Definition 8 (Consistent Observable State). *Let $O = (o^k)_{k\in K}$ be a family of simple observables with value spaces $(V_k)_{k\in K}$, and $(M_i)_{i\in I}$ a family of metamodels. An observable state $(S_i \subseteq \prod_K V_k)_{i\in I}$ is called* consistent *if $\bigcap_{i\in I} S_i \neq \emptyset$.*

Example 7 (Consistency of the EV configuration). The state $\overline{\nu(m)}$ from Ex. 6 is consistent, but would not if $w_{\text{body}} + w_{\text{suspension}} + w_{\text{propulsion}} > w_{\max}$.

The previous definition is essentially the projection of the observable-induced consistency (Def. 2) onto observable states:

Proposition 1. *Let $O = (o^k)_{k\in K}$ be a family of observables. A model $m \in \Pi M$ is consistent for the consistency relation induced by the observables ($m \in CR_O$) iff $\nu(m)$ is admissible and consistent.*

Proof. Directly from the definition of observable-induced consistency (Def. 2).

While Prop. 1 is essentially a sanity check of the construction, it also shifts the perspective on consistency repair: we can recast the goal of finding a consistent model as that of finding a consistent state. To find such a state, we build a transition system via contracts that represent abstract descriptions of how observable states may evolve.

4.2 Contracts

We abstract away from concrete model transformations and consider only their effects on observable values. The motivation is that model transformations modify the structure of models, but their effect on consistency repair can be assessed at the level of observables. More precisely, the abstraction appears at two levels: both on the models and on the transformations. First, since observables encode system-level properties constrained by the models, several models may have the same set of associated observable states, where the exact choice is not relevant for restoring consistency. Second, different model transformations may induce the same change in observable states.

This abstraction leads to the notion of a contract, which specifies transitions between observable states without requiring an explicit model. In this sense, contracts provide an operational abstraction of model evolution that is both semantic and uniform across heterogeneous modeling languages, as they are expressed at the level of observables. Since contracts are meant to be used in a planning process for consistency repair, we define them as relations over observable states.

Notation 2 *We write* $\mathcal{S} = \mathscr{P}\left(\prod_K V_k\right)$*, meaning the observable state space is* $\prod_I \mathcal{S}$*.*

Definition 9 (Contract). *Let* $O = (o^k)_{k\in K}$ *be a family of observables with value spaces* $(V_k)_{k\in K}$ *for an* I*-indexed family of meta-models* $(M_i)_{i\in I}$*.*

A contract *for* O *is a transition relation* $c \subseteq \prod_I \mathcal{S} \times \prod_I \mathcal{S}$*.*

We call the precondition of c*, written* $pre(c)$*, the domain of the relation, i.e.,* $pre(c) = \pi_1(c)$*.*

Example 8 (Contract). Consider a concrete model transformation $c_{\text{suspension}}$ that replaces the suspension model $m_{\text{suspension}}$ by a stricter model $m'_{\text{suspension}}$, while leaving the body and propulsion models unchanged. At the semantic level, this transformation is abstracted by a contract $c_{\text{suspension}} \subseteq \prod_I \mathcal{S} \times \prod_I \mathcal{S}$.

The contract captures only its effect on observable semantics: if $(S, S') \in c_{\text{suspension}}$, then the supported-load observable of the suspension is updated from $[0, L]$ to $[0, L']$ with $L' > L$, while the other observables remain unchanged.

Def. 9 provides a transition system over observable states, in the sense of Def. 5. However, not all observable states are relevant for planning consistency repair: in particular, we can discard all inadmissible states. Indeed, from a model perspective, compound observables are derived, i.e., computed based on the algebra $\mathcal{A}$, so an inadmissible state can never arise as $\nu(m)$ for some model m. Thus, a contract that does not preserve admissibility is not a proper abstraction of a model transformation.

Definition 10 (Admissible Contract). *Let* $\mathcal{S}_{\text{adm}} \subseteq \prod_I \mathcal{S}$ *be the set of admissible states. A contract* c *is* admissible *if*

$$c \subseteq \mathcal{S}_{\text{adm}} \times \mathcal{S}_{\text{adm}} \ .$$

Example 9 (Admissible Contract). The contract $c_{\text{suspension}}$ from Ex. 8 is admissible because replacing the suspension only changes the supported-load observable, while the compound observable w^{total} is still induced from the unchanged component weights.

By contrast, consider a contract that directly replaces the value of the compound observable w^{total} by a smaller one without modifying any of the component-weight observables $o^{\text{body-weight}}$, $o^{\text{motion-weight}}$, or $o^{\text{suspension-weight}}$. The resulting observable state would violate the defining equation of the compound observable, so the contract would not preserve admissibility.

From now on, we silently assume that contracts are always admissible and write it as $c \subseteq \prod_I \mathcal{S} \times \prod_I \mathcal{S}$ under the hypothesis that $c \subseteq \mathcal{S}_{\text{adm}} \times \mathcal{S}_{\text{adm}}$.

Note that model transformations are often *local*, in the sense that they act on a single model rather than on the complete collection of models describing the system (i.e., all components of a V-SUM). At the level of observable states, local contracts operate on a single component S_i, while leaving the other components unchanged. They offer separation of concerns, as their effects do not influence

the observable semantics of other meta-models, meaning they do not influence those models. Locality is therefore a desirable property, which may not always hold. Thus, we introduce it as an additional property rather than including it in the contract definition.

Definition 11 (Local Contract). *A contract c is* local *to a component $i \in I$ if there exists a relation $\mathcal{X}_c \subseteq \mathcal{S} \times \mathcal{S}$ such that for all (S, S'),*

$$(S, S') \in c \iff \big((S_i, S'_i) \in \mathcal{X}_c \ \wedge \ \forall j \neq i,\ S_j = S'_j\big).$$

The notion naturally extends to a subset $J \subseteq I$.

Example 10 (Local Contract). The contract $c_{\text{suspension}}$ from Ex. 8 is local to the suspension component. Indeed, whenever $(S, S') \in c_{\text{suspension}}$, we have $S_{\text{body}} = S'_{\text{body}}$ and $S_{\text{propulsion}} = S'_{\text{propulsion}}$.

Essentially, a local condition encodes a frame condition, meaning that the contract has no side effects. For planning purposes, we also want local contracts not to restrict applicability outside their component(s), i.e., the contract should neither read nor write outside the component i.

A contract encodes a set of transitions on observable states, abstracting away from concrete model transformations. However, there is no guarantee that a contract, defined directly, is indeed the abstraction of any transformation. For the sake of repairing consistency, we want to ensure that whenever we apply a contract, it corresponds to a transition on states that encode the models. We call this property well-formedness.

Definition 12 (Well-formed Contract). *A contract c for O is* well-formed *if $c \subseteq \nu(\Pi M) \times \nu(\Pi M)$.*

Proposition 2. *A well-formed contract is admissible.*

Proof. It follows from Def. 5 and Def. 7 that $\nu(\Pi M) \subseteq \mathcal{S}_{\text{adm}}$.

Example 11 (Well-formed Contract). Assume that the stronger suspension used in $c_{\text{suspension}}$ is an actual component available in the manufacturer's catalogue. Then every target observable state produced by $c_{\text{suspension}}$ is the observable semantics of some suspension model, and the contract is well formed.

A contract allowing the supported load to be increased to an arbitrary value, independently of any realizable suspension model, would not be well formed, since some target observable states would not correspond to any model.

Well-formedness only ensures that both sides of the contract are realizable states. It is a necessary condition for properly abstracting from some model transformations, but it is not sufficient: it might allow one to connect unrelated realizable states.

Definition 13 (Contract Soundness). *Let $T \subseteq \Pi M \times \Pi M$ be a relation on models. A contract c is* sound *with respect to T if for all models $m \in \Pi M$ and all states $S' \in \prod_I \mathcal{S}$, if $(\nu(m), S') \in c$ then there exists $m' \in \Pi M$ such that $(m, m') \in T$ and $\nu(m') = S'$.*

If a contract is sound, every abstract transition corresponds to at least one concrete transformation. Note that a sound contract is well-formed (take the image of T under ν). Soundness connects T, which lies at the engineering layer, with c, which lies at the semantic level, and is visible for planning, explaining how observable-level repairs relate to actual model transformations. Thus, contracts are over-approximations of model transformations.

4.3 Contract Sequences

To find a repair plan, we need to find sequences of contracts that eventually repair consistency. To properly define a sequence of contracts, we need a sequential composition, which is derived from the composition of the relations.

Definition 14 (Contract Composition). *Let c_1 and c_2 be two contracts. Their* composition *is the contract $c_2 \circ c_1$ defined by*

$$\left\{ (S, S'') \in \prod_I \mathcal{S} \times \prod_I \mathcal{S} \mid \exists S' \in \prod_I \mathcal{S},\ (S, S') \in c_1 \wedge (S', S'') \in c_2 \right\} .$$

Admissibility, well-formedness, and soundness are closed under composition, but not locality in the general sense.

Proposition 3. *The following holds:*

1. *The composition of admissible contracts is an admissible contract.*
2. *The composition of local contracts is not a local contract, unless the contracts are local to the same component $i \in I$.*
3. *The composition of well-formed contracts is a well-formed contract.*
4. *The composition of sound contracts is a sound contract.*

Proof. For 1 and 3, the results follow by restriction to $\mathcal{S}_{\text{adm}}$ and $\nu(\Pi M)$ respectively. For 2, closure on the same component follows by restriction to S_i. For 4, closure is obtained by unfolding the existential quantifier.

Since the composition of relations is associative, the composition of contracts also is, meaning that any finite sequence of contracts $(c_1, \ldots, c_n)$ yields a unique composed contract $c_n \circ \cdots \circ c_1$. Based on these, we can define repair plans.

Definition 15 (Repair Plan). *A* repair plan *for an inconsistent observable state $S \in \prod_I \mathcal{S}$ (according to Def. 8) is a finite sequence of contracts $(c_1, \ldots, c_n)$ such that*

1. *S in $pre(c_1)$,*
2. *there exists $S' \in \prod_I \mathcal{S}$, such that $(S, S') \in c_n \circ \cdots \circ c_1$, and*
3. *for all observable states $S' \in \prod_I \mathcal{S}$, if $(S, S') \in c_n \circ \cdots \circ c_1$, then S' is a consistent observable state.*

Example 12 (Repair Plan). Consider states described by $\overline{\nu(m)}$ as in Ex. 6. Suppose that in $S^{(0)}$, we have $w^{(0)}_{\text{body}} = 80$, $w^{(0)}_{\text{propulsion}} = 380$, $w^{(0)}_{\text{suspension}} = 50$, and $w^{(0)}_{\text{max}} = 350$. The computed total vehicle weight exceeds the supported load and $S^{(0)}$ is inconsistent. Assume two admissible contracts: $c_{\text{propulsion}}$, which replaces the propulsion subsystem by a lighter one, and $c_{\text{suspension}}$, which replaces the suspension by one with a higher supported load. Consider the two states $S^{(1)}$ and $S^{(2)}$ induced by $w^{(1)}_{\text{body}} = w^{(2)}_{\text{body}} = 80$, $w^{(1)}_{\text{propulsion}} = w^{(2)}_{\text{propulsion}} = 250$, $w^{(1)}_{\text{suspension}} = w^{(2)}_{\text{suspension}} = 50$, $w^{(1)}_{\text{max}} = 350$, and $w^{(2)}_{\text{max}} = 500$. Assume that $(S^{(0)}, S^{(1)}) \in c_{\text{propulsion}}$, while $(S^{(1)}, S^{(2)}) \in c_{\text{suspension}}$. Note that $S^{(1)}$ is still inconsistent and that $S^{(2)}$ coincides with the consistent observable semantics from Ex. 2. Hence, the sequence

$$S^{(0)} \xrightarrow{c_{\text{propulsion}}} S^{(1)} \xrightarrow{c_{\text{suspension}}} S^{(2)}$$

constitutes a repair sequence for $S^{(0)}$.

The existence of a repair plan for a state corresponding to the observable semantics of a model allows deriving a consistent model, assuming the contracts are sound.

Theorem 1. *Let $T \subseteq \Pi M \times \Pi M$ be a relation on models, $m \notin CR$ be an inconsistent model, and $(c_1, \ldots, c_n)$ be a repair plan for $\nu(m)$. If $(c_1, \ldots, c_n)$ is sound with respect to T, then there exists a sequence of models $(m_0, m_1, \ldots, m_n)$ in ΠM such that*

- *$m_0 = m$,*
- *for all $0 \leq i < n$, $(m_i, m_{i+1}) \in T$,*
- *and $m_n \in CR$.*

Proof. Let $m \in \Pi M$ be such that $m \notin CR$, and let $(c_1, \ldots, c_n)$ be a repair plan for $\nu(m)$. Assume that each c_i is sound with respect to T.

By Def. 15, there exists an observable state $S_n \in \prod_I \mathcal{S}$ such that $(\nu(m), S_n) \in c_n \circ \cdots \circ c_1$. From Def. 14, there exist observable states $(S_0, S_1, \ldots, S_n)$ such that $\nu(m) = S_0$ and for all $0 \leq i < n$, $(S_i, S_{i+1}) \in c_{i+1}$. From Def. 13, we build by induction a sequence of models $(m_0, m_1, \ldots, m_n)$ such that $m_0 = m$, for all $0 \leq i < n$, $(m_i, m_{i+1}) \in T$, and for all $0 \leq i \leq n$, $\nu(m_i) = S_i$. From Def. 15, S_n is consistent, and by Prop. 1, it follows that $m_n \in CR$.

An inconsistent model may admit several repair plans. Additionally, since some contracts may be local to specific components, they do not affect other components and, as such, may be reorganized in the repair plan. Formally, certain contracts commute.

Proposition 4. *The composition of contracts local to disjoint sets of components is commutative.*

Proof. Let c_1 be a contract local to $I_1 \subseteq I$ and c_2 be a contract local to $I_2 \subseteq I$ such that $I_1 \cap I_2 = \emptyset$.

Let $(S, S'') \in c_2 \circ c_1$. From Def. 14, there exists $S' \in \prod_I \mathcal{S}$ such that $(S, S') \in c_1$ and $(S', S'') \in c_2$. By locality, we have for all $k \notin I_1 \cup I_2$, $S_k = S'_k = S''_k$, as well as $S_{i_2} = S'_{i_2}$ for all $i_2 \in I_2$ and $S'_{i_1} = S''_{i_1}$ for all $i_1 \in I_1$.

Now construct $X \in \prod_I \mathcal{S}$ such that $X_{i_2} = S''_{i_2}$ for all $i_2 \in I_2$ and $X_j = S_j$ for $j \notin I_2$. By locality of c_1 and c_2 (the reserve implication), $(X, S'') \in c_1$ and $(S, X) \in c_2$ hold. By Def. 14, that $(S, S'') \in c_1 \circ c_2$, and $c_2 \circ c_1 \subseteq c_1 \circ c_2$.

We obtain the equality by symmetry.

Commutativity has practical implications for repair planning: contracts that are local to disjoint components may be applied in any order. Thus, a repair plan may be reorganized by grouping contracts that act on the same component and permuting those that act on disjoint components. We obtain a canonical way to structure repair plans, separating independent transformations from component-wise refinements. This observation is shown in Section 6, where different orderings of independent contracts lead to the same consistent states.

5 Logical Encoding of Repair Plans

The previous sections formalized observable states and introduced a contract-based view of how observable values can evolve to repair inconsistencies. The next step is operationalization: encoding observable states and contracts into the input languages of modern state-space exploration tools so that these solvers can search for repair plans fully automatically.

The logical encoding should be understood as a symbolic representation of the previously introduced abstract observable transition system. The soundness of the encoding relies on the ability of abstract domains to represent observable value sets faithfully and on the correctness of the logical representation of contracts and consistency predicates.

5.1 Representing Observables

By construction, observable states are set-valued abstractions of the values that an aspect of a system may take. Set-oriented specification languages (B, Z, TLA, etc.) naturally support such abstractions and are well-suited to reasoning about universal, safety-style properties. By contrast, mainstream model checkers and SMT-based tools typically work with first-order, finite, or finitely representable domains. They are optimized for bit-precise or numeric reasoning rather than arbitrary set manipulation. Consequently, a direct translation of the set-theoretic observable model into standard tool input is seldom practical. Instead, we consider three encoding strategies for mapping the inherently set-based state spaces into tool-accessible logical representations:

Set-based Observables are represented directly as set-valued variables: for every observable semantics o_i^k in each component of the V-SUM, a set-valued

variable $S_i^k : \mathscr{P}(V_{o_k})$ is introduced. The encoding is very close to the set-theoretic framework, and consistency conditions can then be formulated directly (cf. Def. 2): for every observable, the intersection of admissible sets across models must be non-empty. While faithful to the original theory [15], this approach relies on the solver being sufficiently expressive to support rich set-manipulation primitives and is often infeasible for scalable analysis.

Value-based Each observable is represented by a single concrete value: a single-value variable $x_i^k : V_o$ is introduced for each observable and model. This reduces the problem to conventional state variables and is well-matched to the feature sets of model-checkers. However, it collapses the set-valued semantics of observables into individual valuations, thereby losing information about admissible alternatives and uncertainty.

Parametric abstract domains As a compromise, observables remain semantically set-valued. However, admissible sets are represented indirectly as finite collections of parameters drawn from suitable abstract domains, which can be encoded in a logical language typically supported by solvers. This bridges the original set-oriented semantics and the value-centric reasoning capabilities of existing tools.

Since the last option enables us to retain the theoretical results established for the set-based setting while still working directly with values, we chose to adopt it in our approach. In the following, we explain how the general abstract search space – where states are sets of sets of values – is refined into a search space over finitely many variables.

5.2 Abstract Domains

We recall the assumed underlying multi-sorted signature $\Sigma = (F_\Sigma, S_\Sigma)$ that we will use to form first-order sentences over Σ.

Definition 16 (Abstract Domain). *An* abstract domain representation *for a value set V is a pair (D, ε_D) of a set D (the parameter space) and a membership relation $\varepsilon_D \subseteq V \times D$. The associated* extent *function $\mu_D : D \to 2^V$ is defined by $\mu_D(d) = \{a \in V \mid a\ \varepsilon_D\ d\}$. Thus, each element $d \in D$ represents the subset $\mu_D(d) \subseteq V$.*

Members of D represent subsets of V and the image $\mu_D(D)$ yields the family of subsets that the abstract domain can represent. For instance, consider the abstract domain $(D_\mathbb{R}, \varepsilon_{D_\mathbb{R}})$ defined by $D_\mathbb{R} := \mathbb{R} \times \mathbb{R}$ and

$$x\ \varepsilon_{D_\mathbb{R}}\ (l, r) \Leftrightarrow l \leq x \leq r\ .$$

Its elements represent closed real intervals via

$$\mu_{D_\mathbb{R}} : D_\mathbb{R} \to 2^\mathbb{R}; \quad (l, r) \mapsto [l, r] \subseteq \mathbb{R}\ .$$

To support symbolic reasoning, we assume that abstract domains admit finite representations over Σ. Concretely, each abstract domain D_k associated with an

observable o^k is assumed to be a finite product $D_k = \prod_{p \in P_k} [\![s_p]\!]_{\mathcal{A}}$, where P_k is a finite index set of parameters and each $s_p \in S_\Sigma$ is a sort. An element of D_k is then a tuple $(d_p)_{p \in P_k}$. Moreover, we require the membership relation ε_{D_k} to be expressible as a first-order formula over Σ. This assumption allows abstract domain elements to be manipulated symbolically within the logical encoding of repair planning.

We say that a simple observable o^k is representable by the abstract domain (D_k, ε_{D_k}) if for for every model instance $m_i \in M_i$, the observable value $o_i^k(m_i)$ belongs to the family of subsets represented by D_k, i.e., if for all $i \in I$ and all $m_i \in M_i$, there exists $d_k \in D_K$ such that $o_i^k(m_i) = \mu_{D_k}(d_k)$.

5.3 Encoding Observables, Contracts and Consistency

Abstract domains enable a symbolic representation of repair planning as a transition system. States encode abstract observable valuations, and transitions encode contract applications. Repair planning then becomes a reachability problem: starting from an inconsistent observable state, find a sequence of contract applications leading to a state satisfying the consistency relation. Formally, we encode the repair problem as a symbolic transition system.

We write $Fml_\Sigma(X)$ for the set of well-formed formulas over Σ with free variables in X. We write $X/\beta \models \varphi$ to indicate that $\varphi \in Fml_\Sigma(X)$ is satisfied under variable assignment β.

Definition 17 (Transition System). *A* transition system $(X, \mathit{Init}, \mathit{Trans})$ *consists of*

- *a finite set X of sorted variables $X = (x_v : s_v)_{v \in V}$ indexed by V, $s_v \in S_\Sigma$,*
- *an initial state formula $\mathit{Init} \in Fml_\Sigma(X)$ and*
- *a transition formula $\mathit{Trans} \in Fml_\Sigma(X \cup X')$ that relates a current state (variables in X) to a successor state (primed copies X').*

For repair planning, the variables in X encode the abstract domain values; the initial formula Init is obtained from the V-SUM instance to be repaired, and the transition formula Trans formalizes the effects of applying a contract. A reachability goal is a formula $\varphi \in Fml_\Sigma(X)$ that characterizes the set of consistent states. A repair plan then corresponds to a finite sequence of assignments $\beta_0, \ldots, \beta_n$ such that $X/\beta_0 \models \mathsf{Init}$, $(X/\beta_i, X'/\beta_{i+1}) \models \mathsf{Trans}$ for $0 \leq i < n$ and $X/\beta_n \models \varphi$. The transition relation Trans encodes the effects of a finite set of contracts on the abstract domains' values.

Encoding Observables The state space for modeling a V-SUM gathers the following set of variables

$$X = \{x_{i,k,p} : s_p \mid i \in I,\ k \in K,\ p \in P_k\},$$

indexed over all meta-models ($i \in I$), all observables ($k \in K$), and all parameters ($p_k \in P_K$) of the abstract domain (D_k, ε_{D_k}). Assignments to the variables in X induce observable states in $\prod_I \mathcal{S}$ via the extent functions μ_{D_k}.

Encoding Initial Conditions For the repair search problem, the variables are initialized by the evaluation of the initial V-SUM $m_I \in \Pi M$. By assumption, each observable value $o_i^k(m_i)$ is representable in the abstract domain D_k. Hence, there exists a tuple of parameter values $(w_{i,k,p} \in [\![s_p]\!]_{\mathcal{A}})_{p \in P_k}$ such that $o_i^k(m_i) = \mu_{D_k}\big((w_{i,k,p})_{p \in P_k}\big)$. We assume that every value $w \in [\![s_p]\!]_{\mathcal{A}}$ can be represented by a ground term $t(w)$ of sort s_p satisfying $[\![t(w)]\!]_{\mathcal{A}} = w$. The initial observable state is then encoded by the formula

$$\mathsf{Init} := \bigwedge_{i \in I} \bigwedge_{k \in K} \bigwedge_{p \in P_k} x_{i,k,p} = t(w_{i,k,p}) \ .$$

Encoding Contracts A contract (Def. 9) is a relation $c \subseteq \prod S \times \prod S$ on the state space. It can be syntactically represented by a before-after-predicate linking the states before and after contract application. Logically, a contract can be formulated as a single two-state formula $C \in Fml_{\Sigma}(X \cup X')$, or, alternatively, as a pre-/postcondition pair $(C^{\text{pre}} \in Fml_{\Sigma}(X), C^{\text{post}} \in Fml_{\Sigma}(X \cup X'))$ following the design-by-contract idea [33]. Each contract formula C represents a relation on abstract domain states (spanned by the state variables in X). Via the extent functions μ_{D_k}, this induces a relation on observable states in $\prod_I \mathcal{S}$, which is required to be a proper representation of the original contract.

We add a synthetic variable $t : \mathbb{N}$ to X to record which contract was applied, so the sequence of applied contracts is recoverable from t's progression.

Def. 11 introduces local contracts that allow modifications within only one component M_i of the V-SUM. Semantically, locality captures the idea that repair actions may be confined to individual components, leaving the rest of the system unchanged. Syntactically, locality is enforced by a *frame condition* stating that variables outside M_i remain unchanged across the transition. Thus, the following frame condition $C^{\text{frame}} \in Fml_{\Sigma}(X \cup X')$ is added:

$$C^{\text{frame}} := \bigwedge_{\substack{j \in I \\ j \neq i}} \bigwedge_{k \in K} \bigwedge_{p \in P_k} x'_{j,k,p} = x_{j,k,p} \ .$$

Like in the case of Def. 11, the locality can be generalized from individual indices $i \in I$ to the more general case of indices $J \subseteq I$ where needed.

The set-theoretic observable states $\mathcal{S}$ also comprise valuations for compound observables whose value set is derived from those of simple observables. One possible encoding strategy is to introduce explicit variables for compound observables, together with constraints ensuring equality between their values and their defining expressions. This option would refine the notion of admissible states (cf. Def. 7) to the abstract domain setting, making compound observables directly accessible within the symbolic state space, but at the cost of additional variables and constraints. Instead, we exploit the fact that compound observable values are already determined implicitly by the variables representing simple observables. Indeed, since we use the same signature Σ for both the definition of compound observables and the logical encoding, values for compound observables can be encoded directly via formulae over the existing state variables in X, avoiding an unnecessary expansion of the symbolic state space.

The transition predicate is then the disjunction of the individual contracts and the bookkeeping of contract applications:

$$\mathsf{Trans} := \bigvee_{x=1}^{n} (C_x^{\text{pre}} \wedge C_x^{\text{post}} \wedge C_x^{\text{frame}} \wedge t' = x) \ ,$$

where the post-state variable t' is used to record which contract was applied.

Encoding Consistency The final ingredient for the reachability encoding is the consistency relation. In this context, we require that the consistency relation must eventually hold. A V-SUM is consistent with respect to the observables if, for each observable, the observables of the individual meta-models share a value:

$$\bigwedge_{k \in K} \left(\exists v : V_k, \ \bigwedge_{i \in I} v \ \varepsilon_{D_k} \ (x_{i,k,p})_{p \in P_k} \right)$$

This formula directly mirrors the original set-theoretic definition of consistency (Def. 2): consistency holds whenever the observable value sets represented by the abstract domains share at least one common semantic value.

The formula requires that the membership predicates ε_{D_k} of the abstract domains used are expressible as formulae. The existential quantifier can often be eliminated. For two intervals in the example abstract domain $(D_{\mathbb{R}}, \varepsilon_{D_{\mathbb{R}}})$ introduced above, the condition $\exists x : \mathbb{R}, x \ \varepsilon_{D_{\mathbb{R}}} \ (l_1, r_1) \wedge x \ \varepsilon_{D_{\mathbb{R}}} \ (l_2, r_2)$ is equivalent to $(l_1 \leq l_2 \leq r_1 \vee l_2 \leq l_1 \leq r_2) \wedge l_1 \leq r_1 \wedge l_2 \leq r_2$.

6 Evaluation

We evaluate the proposed framework using a simple case study that illustrates the interaction among observable-based abstraction, contract-based repair, and logical encoding. The example is designed to exhibit non-trivial dependencies between observables and thus requires the coordinated application of multiple contracts to restore consistency. The set of observables, consistency requirements, and the contracts available to a repair plan are summarised in Table 1. Observables are of three kinds: integer intervals, real intervals, and *dispatched* discrete variables, i.e., variables which select their value from a fixed, predefined domain encoded as a data table. Derived observables correspond to compound observables obtained via the algebraic lifting defined in Section 3.2. The underlying meta-models are abstracted away for simplicity. Experiments were run on an AMD Ryzen AI 7 PRO 360 (16 threads, 30 GiB RAM) under Ubuntu 24.04 with nuXmv 2.1.0.

Following the encoding scheme of Section 5, we translated observables, contracts, and consistency constraints into a transition system in nuXmv. Excerpts from the encoding are shown in Table 2.

We evaluate three invariant-checking algorithms on the same reachability goal: $\neg(\text{step} \leq 10 \wedge \mathit{cons} \wedge \neg \mathit{overflow})$.

(a) Observables

`cell_type`	$\in$ {LFP, NMC, HiVolt}
`body_mat`	$\in$ {steel, alu, comp.}
s (series)	integer $[lo_s, hi_s]$
p (parallel)	integer $[lo_p, hi_p]$
v_b (vol_bod)	real $[lo_{v_b}, hi_{v_b}]$

(b) Consistency relations

`cons_volt`	$300 \le lo\, v_{\text{bat}}, hi\, v_{\text{bat}} \le 450$
`cons_weight`	$hi\, w_{\text{tot}} \le 1500$
`cons_s_rng`	$60 \le lo_s, hi_s \le 120$
`cons_p_rng`	$2 \le lo_p, hi_p \le 12$
`cons_v_rng`	$0.10 \le lo_{v_b}, hi_{v_b} \le 0.30$

(c) Data tables

cell	v_{cell}	w_{cell}	body	ρ_{bod}
LFP	3.2	0.090	steel	7850
NMC	3.5	0.075	alu	4500
HiVolt	3.7	0.060	comp.	2200

(d) Derived observables

v_{bat}	$s \cdot v_{\text{cell}}$
w_{cells}	$s \cdot p \cdot w_{\text{cell}}$
w_{bod}	$v_b \cdot \rho_{\text{bod}}$
w_{tot}	$w_{\text{cells}} + w_{\text{eng}} + w_{\text{bod}}$

(e) Contracts

C1 switch cell	—	$cell' \neq cell$
C2 switch body	—	$body' \neq body$
C3 raise lo_s	$lo_s + 5 \le hi_s$	$lo'_s = lo_s + 5$
C4 lower hi_s	$hi_s - 5 \ge lo_s$	$hi'_s = hi_s - 5$
C5 raise lo_p	$lo_p + 1 \le hi_p$	$lo'_p = lo_p + 1$
C6 lower hi_p	$hi_p - 1 \ge lo_p$	$hi'_p = hi_p - 1$
C7 raise lo_{v_b}	$lo_{v_b} + 0.025 \le hi_{v_b}$	$lo'_{v_b} = lo_{v_b} + 0.025$
C8 lower hi_{v_b}	$hi_{v_b} - 0.025 \ge lo_{v_b}$	$hi'_{v_b} = hi_{v_b} - 0.025$

(f) Initial configuration

`cell_type`	: LFP	$[lo_p, hi_p]$	: $[4, 12]$	$[lo_{v_b}, hi_{v_b}]$	: $[0.20, 0.30]$
`body_mat`	: steel	$[lo_s, hi_s]$	: $[60, 120]$	violated	: `cons_volt`, `cons_weight`

Table 1: Setup of the evaluation case.

IC3 [10] performs property-directed reachability. It builds an inductive certificate by stepping backward from the property and returns a counterexample when the certificate cannot be closed. **BMC** [1] searches for bounded counterexamples using SMT. It unrolls the transition relation up to a bounded depth and asks an SMT solver for a satisfying trace. The *falsification* flag instructs nuXmv to halt as soon as a violating trace is found rather than continuing to prove the bound safe. **Incremental COI** [9, 11] is a nuXmv-specific BMC variant. It restricts each unrolling to the property's cone of influence and reuses solver state across bounds. All three algorithms returned the same set of 7 moves, in different orders. Table 3 shows the COI ordering.

The resulting repair plan consists of seven contract applications. The plan combines structural changes (C1, C2) with successive refinements of interval bounds (C3). The repeated application of C3 illustrates how contracts can be composed to adjust observable values progressively. Note that contracts C1 and

(a) Observable encoding

```
VAR
 cell_type    : 1..3;
 body_mat     : 1..3;
 lo_series    : 1..200;
 hi_series    : 1..200;
 lo_vol_bod   : real;
 hi_vol_bod   : real;
 ...
```

(b) One contract (C3: raise lo_s)

```
pre_3  := lo_series + 5
 <= hi_series;
post_3 := next(lo_series)
 = lo_series + 5;
frame_lo_series := {3};
...
```

(c) Initial state

```
INIT
 cell_type    = 1 &
 body_mat     = 1 &
 lo_series    = 60 &
 hi_series    = 120 &
 lo_parallel  = 4 &
 hi_parallel  = 12 &
 lo_vol_bod   = 0.20 &
 hi_vol_bod   = 0.30
 ...
```

(d) Transition relation

```
TRANS
 ((pre_1 & post_1
 & next(t)=1) |
 (pre_2 & post_2
 & next(t)=2) |
 ...
 ) &
 (!(next(t) in
 frame_cell_type) ->
 next(cell_type)
 = cell_type) &
 ...
```

Table 2: Excerpts from the nuXmv encoding (truncated for space).

C2 are local to distinct components and therefore commute with the interval adjustments, which explains why the different algorithms return equivalent plans up to reordering, as formally justified by Prop. 4.

It is important to note that repair actions operate on observable states rather than directly on design-variable assignments. A repair may therefore restrict an observable set, narrowing the admissible ranges. The objective is not to maximize the size of the parametric sets, but restore overlap between the sets of the observable state. Hence, in order to restore consistency, contracts may decrease the cardinalities of sets – but add relevant values that ensure overlap. In particular, contracts that narrows admissible ranges should not be interpreted as reducing the set of feasible system designs.

Step	Contract	Change
1	C3	raise lo_s: 60 → 65
2	C1	switch cell: LFP → HiVolt
3	C3	raise lo_s: 65 → 70
4	C3	raise lo_s: 70 → 75
5	C2	switch body: steel → composite
6	C3	raise lo_s: 75 → 80
7	C3	raise lo_s: 80 → 85

Table 3: Repair plan returned by COI.

Table 4 compares per-algorithm runtimes. This case study is not intended as a comprehensive benchmark, but rather to show the feasibility of the approach. It also provides a hint about the impact of the different strategies: incremental COI seems to benefit from restricting the search space to relevant variables, and results in a lower runtime than both IC3 and plain BMC.

	IC3	BMC	COI
length	7	7	7
wall time	16.6 s	18.6 s	12.7 s
peak mem	369 M	149 M	197 M

Table 4: Solver runtimes on the same goal property.

Overall, the case study shows that the proposed framework enables the automatic synthesis of repair plans from a purely observable-based specification. The results confirm that the logical encoding preserves the structure of contracts and their composition, and that standard state-space exploration techniques can effectively compute repair sequences satisfying the consistency requirements.

7 Discussion

Our approach was motivated by separating concerns across several layers: observables (semantic abstraction of the model via system-level properties), consistency (relation over observable values), contracts (admissible transformations on these observables), and planning (sequences of contracts that restore consistency). These different layers naturally allow for some design trade-offs. For instance, observables can be chosen to reflect domain-relevant properties at various levels of abstraction. Similarly, consistency requirements can be made to evolve independently from the operational mechanisms used to restore them, as advocated in [15].

While contracts provide an operational approximation of model transformations, they may not encode all possible model changes. They should be sound in the sense that each contract step should correspond to a valid model transformation to ensure the correctness of our method. This correspondence is a modeling concern that may require validating or refining contracts, especially for abstract domains.

Since contracts act on observable states, they also induce a trade-off between modeling effort and planning complexity. Complex observables simplify consistency checks and reduce planning complexity, but may be harder to encode and manipulate. Abstract domains provide a compromise by requiring tractable set-based observable semantics with solver-friendly representations, at the cost of potential approximations.

Since contracts that act on disjoint components commute, contract locality can be used to control the structure of repair plans. More precisely, some steps can be swapped without changing the result. This explains why different solvers return plans that are similar up to ordering. From a planning perspective, locality also justifies preferring canonical or minimal representatives for repair plans.

The objective of our approach is to synthesise repair plans that provide traceability from the inconsistent state to the repaired state, rather than solely producing a witness of a consistent target state. This traceability is of particular relevance in engineering contexts, where model evolution must remain auditable: the generated repair plans document which transformations are to be applied and in which order, thereby rendering consistency repair a controlled evolutionary process. Given the centrality of repair plans to our approach, the repair problem cannot be directly encoded as a SAT or SMT problem, as would otherwise be possible.

8 Related Work

MDD faces the challenge of maintaining consistency between multiple overlapping descriptions of a system. Some approaches address this challenge by introducing ground truth encoded as a single unified model or *single underlying model (SUM)* [2,3]. All domain-specific models are then derived from a SUM as *views*, i.e., projections. While conceptually simple, this approach suffers from scalability and maintenance issues. Approaches such as the virtual single underlying models [25] overcome these limitations by distributing system representation to multiple interlinked models. The connection between models is defined by consistency preservation rules that describe how changes in one model are propagated to other models. More recently, Pascual et al. [34] proposed formal foundations for V-SUMs and developed a semantic notion for model consistency. Consistency is characterized at the level of semantic domains and induces a structured lattice of model relations. These foundations provide a basis for reasoning about consistency beyond syntactic correspondences and a natural playground for principled approaches to repair consistency.

Model Repair Techniques Model repair approaches aim at restoring consistency after violations, either within a single model or across multiple models. They often take the shape of *controlled updates* propagating changes in one model into other models to preserve consistency. Examples of controlled updates include change propagation between multiple architectural languages [14], *bidirectional transformations* that automatically synchronize and update models [13, 31, 39], as well as change propagation in general [20]. In these settings, consistency is enforced by construction, disallowing temporary inconsistencies.

More flexible approaches allow user-defined consistency constraints and corresponding repair strategies. While often meta-model-independent, such approaches generally rely on the definition of *repair hints*, i.e., rules that guide the repair process for each constraint [12, 26, 37]. The definition of repair hints is often

labor-intensive and error-prone, as it requires expertise from multiple domains, making them difficult to scale in heterogeneous settings.

Another line of work relies on graph transformations, where models are represented as graphs and consistency is expressed via graph predicates or triple graph grammars. Consistency repair is then obtained by applying rules that restore the violated constraints. Kosiol et al. [27, 28] introduce *graduated graph consistency* along with analyses to ensure the preservation or improvement of consistency levels. Fritsche et al. [17, 18] propose higher-order rules and triple graph grammars for concurrent synchronization. While these approaches provide strong formal guarantees, they rely on explicit rules, i.e., at the graph level, which can be hard to obtain for heterogeneous models or when consistency is defined at the semantic level.

Several approaches based on artificial intelligence have also been proposed for model repair [4]. These methods can reduce the need for manually specified repair strategies, but typically depend on training data. When the consistency constraints between models are user-defined and project-specific, it remains unclear how well training data transfers across development contexts.

There are also search-based model repair approaches. A notable example is Badger, a regression planner that uses consistency preservation rules [35, 36]. It assumes predefined inconsistency types, detected via pattern matching and repaired via associated rules. However, Badger considers only intra-model inconsistencies.

In contrast to these approaches, we consider consistency at the level of observable semantics and formulate repair as a problem over abstract observable states. Thus, we shift the focus from model-level repair rules to a separation between (i) observable abstractions that capture semantic properties and (ii) contracts that describe admissible transformations. Thus, the approach avoids the need for constraint-specific repair hints, while supporting heterogeneous models and semantically defined consistency relations. Repair is then operationalized as a planning problem over the resulting transition system.

Planning A planning problem can be defined as the task of finding a sequence of actions in a transition system that leads from a given initial state to a specified goal state. Classical approaches are based on STRIPS [16], with later work focusing on propositional encodings to reduce the complexity of the search space. Kautz et al. [24] proposed a satisfiability-based formulation of planning, meaning that planning problems can be encoded as SAT instances. Subsequent work demonstrated the effectiveness of SAT, SMT, and constraint programming techniques for solving planning problems [5,6,8,23,29,38], as well as extensions such as Planning Modulo Theories [19].

These approaches provide the algorithmic foundation for exploring sequences of transformations that we use to plan consistency repairs. This connection enables the use of off-the-shelf verification and planning tools for automated repair synthesis.

Contracts Note that while we use contracts for the design of cyber-physical systems, our approach differs from standard approaches of contracts system design as proposed by Benveniste et al. [7] or as used in Pacti [21]. In the meta-theory of contracts, Benveniste et al. [7, Chap. 4] define contract semantics as a set of implementations and a set of environments (both understood as subsets of all possible components). Assume/Guarantee contracts are then derived as a special case [7, Chap. 5], further refined in [21] via a universe of behaviors. These behavior rely on all variables of the system, in a similar fashion as our approach based on observables [15]. However, Assume/Guarantee contracts apply to components, i.e., models in our approach, while we design contracts as an abstraction of model transformation, i.e., changes on changes on components in the sense of Benveniste et al.

9 Conclusion

We presented a method to repair inter-model inconsistencies by reducing model consistency repair to planning in a shared space of observable semantics. By encoding repairs as sequences of contract applications over observable variables, we decouple domain-specific semantic notions from the repair search, enabling solver-based exploration. Our method synthesizes a sequence of admissible consistency repair actions instead of searching for the consistent target state directly. This provides traceability from the initial inconsistency to the repaired system state which allows model evolution to remain controlled and auditable. Our approach is a first step toward tolerating temporary inconsistencies in a distributed model-driven design environment. Future work will investigate the design of contract libraries, improve encodings for SMT/model checkers, and evaluate scalability and solver trade-offs.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – CRC 1608 – 501798263.

References

1. Armando, A., Mantovani, J., Platania, L.: Bounded model checking of software using SMT solvers instead of SAT solvers. International Journal on Software Tools for Technology Transfer **11**(1), 69–83 (2009). https://doi.org/10.1007/s10009-008-0091-0
2. Atkinson, C., Bostan, P., Brenner, D., Falcone, G., Gutheil, M., Hummel, O., Juhasz, M., Stoll, D.: Modeling Components and Component-Based Systems in KobrA. In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.) The Common Component Modeling Example: Comparing Software Component Models, pp. 54–84. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85289-6_4
3. Atkinson, C., Stoll, D., Bostan, P.: Orthographic Software Modeling: A Practical Approach to View-Based Development. In: Maciaszek, L.A., González-Pérez,

C., Jablonski, S. (eds.) Evaluation of Novel Approaches to Software Engineering. pp. 206–219. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14819-4_15
4. Barriga, A., Rutle, A., Heldal, R.: AI-powered model repair: An experience report—lessons learned, challenges, and opportunities. Software and Systems Modeling **21**(3), 1135–1157 (Jun 2022). https://doi.org/10.1007/s10270-022-00983-5
5. Barták, R., Toropila, D.: Reformulating Constraint Models for Classical Planning. In: Wilson, D., Lane, H.C. (eds.) Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA. pp. 525–530. AAAI Press (2008)
6. van Beek, P., Chen, X.: CPlan: A Constraint Programming Approach to Planning. In: Hendler, J., Subramanian, D. (eds.) Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA. pp. 585–590. AAAI Press / The MIT Press (1999)
7. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T.A., Larsen, K.G.: Contracts for system design. Foundations and Trends in Electronic Design Automation **12**(2-3), 124–400 (2018). https://doi.org/10.1561/1000000053
8. Bofill, M., Espasa, J., Villaret, M.: The RANTANPLAN planner: System description. The Knowledge Engineering Review **31**(5), 452–464 (Nov 2016). https://doi.org/10.1017/S0269888916000229
9. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: Computer Aided Verification (CAV 2014). LNCS, vol. 8559, pp. 334–342. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
10. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Computer Aided Verification (CAV 2012). Lecture Notes in Computer Science, vol. 7358, pp. 277–293. Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_23
11. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
12. Easterbrook, S., Nuseibeh, B.: Using ViewPoints for inconsistency management. Software Engineering Journal **11**(1), 31–43 (Jan 1996). https://doi.org/10.1049/sej.1996.0004
13. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information Preserving Bidirectional Model Transformations. In: Dwyer, M.B., Lopes, A. (eds.) Fundamental Approaches to Software Engineering. pp. 72–86. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71289-3_7
14. Eramo, R., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: A model-driven approach to automate the propagation of changes among Architecture Description Languages. Software & Systems Modeling **11**(1), 29–53 (Feb 2012). https://doi.org/10.1007/s10270-010-0170-z
15. Färber, H., Pascual, R., Stübinger, T., Ulbrich, M.: Observable semantics for characterising consistency between heterogeneous models. In: Bianculli, D., Gómez-Martínez, E. (eds.) Software Engineering and Formal Methods. pp. 110–128. Springer Nature Switzerland, Cham (2025)
16. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3-4), 189–208 (Dec 1971). https://doi.org/10.1016/0004-3702(71)90010-5

17. Fritsche, L., Kosiol, J., Möller, A., Schürr, A.: Advanced Consistency Restoration with Higher-Order Short-Cut Rules. In: Graph Transformation. pp. 184–203. Lecture Notes in Computer Science (2023). https://doi.org/10.1007/978-3-031-36709-0_10
18. Fritsche, L., Kosiol, J., Möller, A., Schürr, A., Taentzer, G.: A precedence-driven approach for concurrent model synchronization scenarios using triple graph grammars. In: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering. pp. 39–55. SLE 2020, Association for Computing Machinery, New York, NY, USA (Nov 2020). https://doi.org/10.1145/3426425.3426931
19. Gregory, P., Long, D., Fox, M., Beck, J.C.: Planning Modulo Theories: Extending the Planning Paradigm. Proceedings of the International Conference on Automated Planning and Scheduling **22**, 65–73 (May 2012). https://doi.org/10.1609/icaps.v22i1.13505
20. Hinkel, G.: Change Propagation in an Internal Model Transformation Language. In: Kolovos, D., Wimmer, M. (eds.) Theory and Practice of Model Transformations. pp. 3–17. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21155-8_1
21. Incer, I., Badithela, A., Graebener, J.B., Mallozzi, P., Pandey, A., Rouquette, N., Yu, S.J., Benveniste, A., Caillaud, B., Murray, R.M., Sangiovanni-Vincentelli, A., Seshia, S.A.: Pacti: Assume-Guarantee Contracts for Efficient Compositional Analysis and Design. ACM Trans. Cyber-Phys. Syst. **9**(1), 3:1–3:35 (Jan 2025). https://doi.org/10.1145/3704736
22. Kamburjan, E., Fiorini, S.R.: On the Notion of Naturalness in Formal Modeling. In: Ahrendt, W., Beckert, B., Bubel, R., Johnsen, E.B. (eds.) The Logic of Software. A Tasting Menu of Formal Methods: Essays Dedicated to Reiner Hähnle on the Occasion of His 60th Birthday, pp. 264–289. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-08166-8_13, https://doi.org/10.1007/978-3-031-08166-8_13
23. Kautz, H.A.: Deconstructing Planning as Satisfiability. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA. pp. 1524–1526. AAAI Press (2006)
24. Kautz, H.A., Selman, B.: Planning as Satisfiability. In: Neumann, B. (ed.) 10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings. pp. 359–363. John Wiley and Sons (1992)
25. Klare, H., Kramer, M.E., Langhammer, M., Werle, D., Burger, E., Reussner, R.: Enabling consistency in view-based system development — The Vitruvius approach. Journal of Systems and Software **171**, 110815 (Jan 2021). https://doi.org/10.1016/j.jss.2020.110815
26. Kolovos, D., Paige, R., Polack, F.: Detecting and Repairing Inconsistencies across Heterogeneous Models. In: And Validation 2008 1st International Conference on Software Testing, Verification. pp. 356–364 (Apr 2008). https://doi.org/10.1109/ICST.2008.23
27. Kosiol, J., Strüber, D., Taentzer, G., Zschaler, S.: Graph Consistency as a Graduated Property. In: Gadducci, F., Kehrer, T. (eds.) Graph Transformation. pp. 239–256. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51372-6_14
28. Kosiol, J., Strüber, D., Taentzer, G., Zschaler, S.: Sustaining and improving graduated graph consistency: A static analysis of graph transformations. Science of

Computer Programming **214**, 102729 (Feb 2022). https://doi.org/10.1016/j.scico.2021.102729
29. Leofante, F., Giunchiglia, E., Ábráham, E., Tacchella, A.: Optimal Planning Modulo Theories. In: Twenty-Ninth International Joint Conference on Artificial Intelligence. vol. 4, pp. 4128–4134 (Jul 2020). https://doi.org/10.24963/ijcai.2020/571
30. Liu, T., Tyszberowicz, S., Beckert, B.: Observable Consistency Checking across Requirements and Models. In: 21st International Conference on Evaluation of Novel Approaches to Software Engineering. pp. 50–63 (Jun 2026)
31. Macedo, N., Cunha, A.: Least-change bidirectional model transformation with QVT-R and ATL. Software & Systems Modeling **15**(3), 783–810 (Jul 2016). https://doi.org/10.1007/s10270-014-0437-x
32. Macedo, N., Jorge, T., Cunha, A.: A Feature-Based Classification of Model Repair Approaches. IEEE Transactions on Software Engineering **43**(7), 615–640 (Jul 2017). https://doi.org/10.1109/TSE.2016.2620145
33. Meyer, B.: Applying "design by contract". Computer **25**(10) (1992)
34. Pascual, R., Beckert, B., Ulbrich, M., Kirsten, M., Pfeifer, W.: Formal Foundations of Consistency in Model-Driven Development. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Specification and Verification (ISOLA 2024). pp. 178–200. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-75380-0_11
35. Pinna Puissant, J., Van Der Straeten, R., Mens, T.: Badger: A Regression Planner to Resolve Design Model Inconsistencies. In: Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D. (eds.) Modelling Foundations and Applications. pp. 146–161. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31491-9_13
36. Pinna Puissant, J., Van Der Straeten, R., Mens, T.: Resolving model inconsistencies using automated regression planning. Software & Systems Modeling **14**(1), 461–481 (Feb 2015). https://doi.org/10.1007/s10270-013-0317-9
37. Reder, A., Egyed, A.: Computing repair trees for resolving inconsistencies in design models. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 220–229. ASE '12, Association for Computing Machinery, New York, NY, USA (Sep 2012). https://doi.org/10.1145/2351676.2351707
38. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: Parallel plans and algorithms for plan search. Artificial Intelligence **170**(12), 1031–1080 (Sep 2006). https://doi.org/10.1016/j.artint.2006.08.002
39. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software & Systems Modeling **12**(1), 89–104 (Feb 2013). https://doi.org/10.1007/s10270-010-0187-3