Towards the Automated Generation of Consistent, Diverse, Scalable and Realistic Graph Models

Dániel Varró[1,2,3], Oszkár Semeráth[1,2], Gábor Szárnyas[1,2], and
Ákos Horváth[1,4]

[1] Budapest University of Technology and Economics
[2] MTA-BME Lendület Research Group on Cyber-Physical Systems
[3] McGill University, Department of Electrical and Computer Engineering
[4] IncQuery Labs Ltd.
{varro,semerath,szarnyas,ahorvath}@mit.bme.hu

Abstract. Automated model generation can be highly beneficial for various application scenarios including software tool certification, validation of cyber-physical systems or benchmarking graph databases to avoid tedious manual synthesis of models. In the paper, we present a long-term research challenge how to generate graph models specific to a domain which are consistent, diverse, scalable and realistic at the same time.

We provide foundations for a class of model generators along a refinement relation which operates over partial models with 3-valued representation and ensures that subsequently derived partial models preserve the truth evaluation of well-formedness constraints in the domain. We formally prove *completeness*, i.e. any finite instance model of a domain can be generated by model generator transformations in finite steps and *soundness*, i.e. any instance model retrieved as a solution satisfies all well-formedness constraints. An experimental evaluation is carried out in the context of a statechart modeling tool to evaluate the trade-off between different characteristics of model generators.

Keywords: automated model generation, partial models, refinement

1 Introduction

Smart and safe cyber-physical systems [16, 54, 69, 93] are software-intensive autonomous systems that largely depend on the context in which they operate, and frequently rely upon intelligent algorithms to adapt to new contexts on-the-fly. However, adaptive techniques are currently avoided in many safety-critical systems due to major certification issues. Automated synthesis of prototypical test contexts [58] aims to systematically derive previously unanticipated contexts for assurance of such smart systems in the form of graph models. Such prototype contexts need to be *consistent*, i.e. they need to fulfill certain well-formedness (consistency) constraints when synthesizing large and realistic environments.

In many design and verification tools used for engineering CPSs, system models are frequently represented as typed and attributed graphs. There has been an increasing interest in model generators to be used for validating, testing or benchmarking design tools with advanced support for queries and transformations [4, 6, 42, 92]. Qualification of design and verification tools is necessitated by safety standards (like DO-178C [89], or ISO 26262 [43]) in order to assure that their output results can be trusted in safety-critical applications. However, tool qualification is extremely costly due to the lack of effective best practices for validating the design tools themselves. Additionally, design-space exploration [47,57,66] necessitates to automatically derive different solution candidates which are optimal wrt. certain objectives for complex allocation problems. For testing and DSE purposes, *diverse* models need to be synthesized where any pairs of models are structurally very different from each other in order to achieve high coverage or a diverse solution space.

Outside the systems engineering domain, many performance benchmarks for advanced relational databases [26], triple stores and graph databases [13,60,80], or biochemical applications [36,99] also rely on the availability of extremely large and *scalable* generators of graph models.

Since real models created by engineers are frequently unavailable due to the protection of intellectual property rights, there is an increasing need of *realistic* models which have similar characteristics to real models. However, these models should be domain-specific, i.e. graphs of biomedical systems are expected to be very different from graphs of social networks or software models. An engineer can easily distinguish an auto-generated model from a manually designed model by inspecting key attributes (e.g. names), but the same task becomes more challenging if we abstract from all attributes and inspect only the (typed) graph structure. While several graph metrics have been proposed [10, 12, 44, 68], the characterization of *realistic* models is a major challenge [91].

As a long-term research challenge, we aim at automatically generating domain-specific graph models which are simultaneously scalable, realistic, consistent and diverse. In the paper, we precisely formulate the model generation challenge for the first time (Section 2). Then in Section 3, we revisit the formal foundations of partial models and well-formedness constraints captured by graph patterns. In Section 4, we propose a refinement calculus for partial models as theoretical foundation for graph model generation, and a set of specific refinement operations as novel contributions. Moreover, we precisely formulate certain soundness and completeness properties of this refinement calculus.[5] In addition, we carry out an experimental evaluation of some existing techniques and tools in Section 5 to assess the trade-off between different characteristics (e.g. diverse vs. realistic, consistent vs. diverse, diverse vs. consistent and consistent vs. scalable) of model generation. Finally, related work is discussed wrt. the different properties required for model generation in Section 6.

---

[5] The authors' copy of this paper is available at `https://inf.mit.bme.hu/research/publications/towards-model-generation` together with the proofs of theorems presented in Section 4.

2 The Graph Model Generation Challenge

A domain specification (or domain-specific language, DSL) is defined by a *meta-model MM* which captures the main concepts and relations in a domain, and specifies the basic graph structure of the models. In addition, a set of *well-formedness constraints* $WF = \{\phi_1, \ldots, \phi_n\}$ may further restrict valid domain models by extra structural restrictions. Furthermore, we assume that *editing operations* of the domain are also defined by a set of rules OP.

Informally, the *automated model generation challenge* is to derive a set of instance models where each M_i conforms to a metamodel MM. A model generator $Gen \mapsto \{M_i\}$ derives a set (or sequence) of models along a derivation sequence $M_0 \xrightarrow{op_1,\ldots,op_k} M_i$ starting from (a potentially empty) initial model M_0 by applying some operations op_j from OP at each step. Ideally, a single model M_i or a model generator Gen should satisfy the following requirements:

- **Consistent (CON)**: A model M_i is consistent if it satisfies all constraints in WF (denoted by $M_i \models WF$). A model generator Gen is consistent, if it is sound (i.e. if a model is derivable then it is consistent) and complete (i.e. all consistent models can be derived).
- **Diverse (DIV)**: The diversity of a model M_i is defined as the number of (direct) types used from its MM: M_i is more diverse than M_j if more types of MM are used in M_i than in M_j. A model generator Gen is diverse if there is a designated distance between each pairs of models M_i and M_j: $dist(M_i, M_j) > D$.
- **Scalable (SCA)**: A model generator Gen is scalable *in size* if the size of M_i is increasing exponentially $\#(M_{i+1}) \geq 2 \cdot \#(M_i)$, thus a single model M_i can be larger than a designated model size $\#(M_i) > S$. A model generator Gen is scalable *in quantity* if the generation of M_j (of similar size) does not take significantly longer than the generation of any previous model M_i: $time(M_j) < max_{0 \leq i < j}\{time(M_i)\} \cdot T$ (for some constant T).
- **Realistic (REA)**: A generated model is (structurally) realistic if it cannot be distinguished from the structure of a real model after all text and values are removed (by considering them irrelevant). A model generator is realistic wrt. some graph metrics [91] and a set of real models $\{RM_i\}$ if the evaluation of the metrics for the real and the generated set of models has similar values: $\big|metr(\{RM_i\}) - metr(\{M_i\})\big| < R$.

Note that we intentionally leave some metrics *metr* and distance functions *dist* open in the current paper as their precise definitions may either be domain-specific or there are no guidelines which ones are beneficial in practice.

Each property above is interesting in itself, i.e. it has been addressed in numerous papers, and used in at least one industrial application scenario. Moreover, similar properties might be defined in the future. However, the grand challenge is to develop an automated model generator which simultaneously satisfies multiple (ideally, all four) properties. For instance, a model generator for benchmarking purposes needs to be scalable, realistic and consistent, while a test model generator needs to be diverse, consistent (or intentionally faulty), and scalable in

quantity. However, existing model generation approaches developed in different research areas usually support one (or rarely at most two) of these properties.

Such a multi-purpose model generator is out of scope also for the current paper. In fact, as a novel contribution, we provide precise theoretical foundations for a graph model generator that is scalable and consistent based on a refinement calculus. Our specific focus is motivated by a novel empirical evaluation to be reported in Section 5 which states that consistency is a prerequisite for the synthesis of both diverse and realistic models.

3 Preliminaries

We illustrate automated model generation in the context of Yakindu Statecharts Tools [101], which is an industrial DSL developed by Itemis AG for the development of reactive, event-driven systems using statecharts captured in a combined graphical and textual syntax. Yakindu supports validation of WF constraints, simulation and code generation from statechart models. We first revisit the formalization of the partial models and WF-constraints as defined in [85].

3.1 Metamodels and instance models

Formally, a *metamodel* defines a vocabulary $\Sigma = \{\texttt{C}_1, \ldots, \texttt{C}_n, \texttt{R}_1, \ldots, \texttt{R}_m, \sim\}$ where a unary predicate symbol $\texttt{C}_i$ ($1 \leq i \leq n$) is defined for each class (node type), and a binary predicate symbol $\texttt{R}_j$ ($1 \leq j \leq m$) is defined for each reference (edge type). The index of a predicate symbol refers to the corresponding metamodel element. The binary $\sim$ predicate is defined as an equivalence relation over objects (nodes) to denote if two objects can be merged. For space considerations, we omit the precise handling of attributes from this paper as none of the four key properties depend on attributes. For metamodels, we use the notations of the Eclipse Modeling Framework (EMF) [90], but our concepts could easily be adapted to other frameworks of typed and attributed graphs such as [21, 28].

An *instance model* is a 2-valued logic structure $M = \langle Obj_M, \mathcal{I}_M \rangle$ over Σ where $Obj_M = \{o_1, \ldots, o_n\}$ ($n \in \mathbb{Z}^+$) is a finite set of individuals (objects) in the model (where $\#(M) = |Obj_M| = n$ denotes the size of the model) and $\mathcal{I}_M$ is a 2-valued interpretation of predicate symbols in Σ defined as follows (where o_k and o_l are objects from Obj_M with $1 \leq k, l \leq n$):

- **Type predicates:** the 2-valued interpretation of a predicate symbol $\texttt{C}_i$ in M (formally, $\mathcal{I}_M(\texttt{C}_i) : Obj_M \to \{1, 0\}$) evaluates to 1 if object o_k is instance of class $\texttt{C}_i$ (denoted by $[\![\texttt{C}_i(o_k)]\!]^M = 1$), and evaluates to 0 otherwise.
- **Reference predicates:** the 2-valued interpretation of a predicate symbol $\texttt{R}_j$ in M (formally, $\mathcal{I}_M(\texttt{R}_j) : Obj_M \times Obj_M \to \{1, 0\}$) evaluates to 1 if there exists an edge (link) of type $\texttt{R}_j$ from o_k to o_l in M denoted as $[\![\texttt{R}_j(o_k, o_l)]\!]^M = 1$, and evaluates to 0 otherwise.
- **Equivalence predicate:** the 2-valued interpretation of a predicate symbol $\sim$ in M (formally, $\mathcal{I}_M(\sim) : Obj_M \times Obj_M \to \{1, 0\}$) evaluates to 1 for any

object o_k, i.e. $[\![o_k \sim o_k]\!]^M = 1$, and evaluates to 0 for any different pairs of objects, i.e. $[\![o_k \sim o_l]\!]^M = 0$, if $o_k \neq o_l$. This equivalence predicate is rather trivial for instance models but it will be more relevant for partial models.

3.2 Partial models

Partial models [31,46] represent uncertain (possible) elements in instance models, where one partial model represents a set of concrete instance models. In this paper, 3-valued logic [48] is used to explicitly represent unspecified or unknown properties of graph models with a third $1/2$ value (beside 1 and 0 which stand for *true* and *false*) in accordance with [76,85].

A *partial model* is a 3-valued logic structure $P = \langle Obj_P, \mathcal{I}_P \rangle$ of Σ where $Obj_P = \{o_1, \ldots, o_n\}$ ($n \in \mathbb{Z}^+$) is a finite set of individuals (objects) in the model, and $\mathcal{I}_P$ is a 3-valued interpretation for all predicate symbols in Σ defined below. The 3-valued truth evaluation of the predicates in a partial model P will be denoted respectively as $[\![\mathsf{C}_i(o_k)]\!]^P$, $[\![\mathsf{R}_j(o_k, o_l)]\!]^P$, $[\![o_k \sim o_l]\!]^P$.

- **Type predicates:** $\mathcal{I}_P$ gives a 3-valued interpretation for each class symbol C_i in Σ: $\mathcal{I}_P(\mathsf{C}_i) : Obj_P \to \{1, 0, 1/2\}$, where 1, 0 and $1/2$ means that it is true, false or unspecified whether an object is an instance of a class C_i.
- **Reference predicates:** $\mathcal{I}_P$ gives a 3-valued interpretation for each reference symbol R_j in Σ: $\mathcal{I}_P(\mathsf{R}_j) : Obj_P \times Obj_P \to \{1, 0, 1/2\}$, where 1, 0 and $1/2$ means that it is true, false or unspecified whether there is a reference of type R_j between two objects.
- **Equivalence predicate:** $\mathcal{I}_P$ gives a 3-valued interpretation for the $\sim$ relation between the objects $\mathcal{I}_P(\sim) : Obj_P \times Obj_P \to \{1, 0, 1/2\}$.
  A predicate $o_k \sim o_l$ between two objects o_k and o_l is interpreted as follows:
  - If $[\![o_k \sim o_l]\!]^P = 1$ then o_k and o_l are equal and they can be merged;
  - If $[\![o_k \sim o_l]\!]^P = 1/2$ then o_k and o_l may be equal and may be merged;
  - If $[\![o_k \sim o_l]\!]^P = 0$ then o_k and o_l are different objects in the instance model, thus they cannot be merged.
  A predicate $o_k \sim o_k$ for any object o_k (as a self-edge) means the following:
  - If $[\![o_k \sim o_k]\!]^P = 1$ then o_k is a final object which cannot be further split to multiple objects;
  - If $[\![o_k \sim o_k]\!]^P = 1/2$ then o_k is a multi-object which may represent a set of objects.
  The traditional properties of the equivalence relation $\sim$ are interpreted as:
  - $\sim$ is a symmetric relation: $[\![o_k \sim o_l]\!]^P = [\![o_l \sim o_k]\!]^P$;
  - $\sim$ is a reflexive relation: $[\![o_k \sim o_k]\!]^P > 0$;
  - $\sim$ is a transitive relation: $[\![o_k \sim o_l \wedge o_l \sim o_m \Rightarrow o_k \sim o_m]\!]^P > 0$ which prevents that $[\![o_k \sim o_l]\!]^P = 1$, $[\![o_l \sim o_m]\!]^P = 1$ but $[\![o_l \sim o_m]\!]^P = 0$.

Informally, this definition of partial models is very general, i.e. it does not impose any further restriction imposed by a particular underlying metamodeling technique. For instance, in case of EMF, each object may have a single direct type
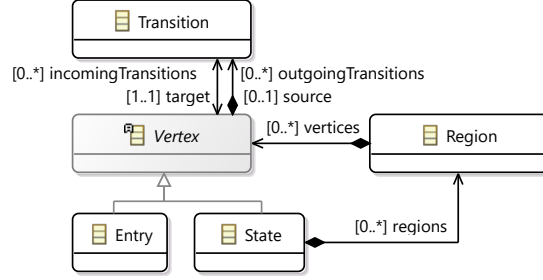
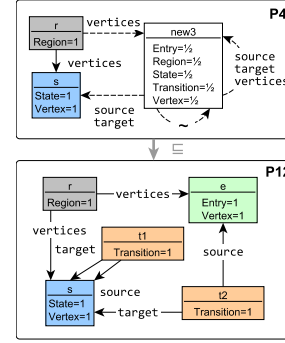Fig. 1: Metamodel extract of Yakindu statecharts

Fig. 2: Partial models

and needs to be arranged into a strict containment hierarchy while graphs of the semantic web may be flat and nodes may have multiple types. Such restrictions will be introduced later as structural constraints. Mathematically, partial models show close resemblance with graph shapes [75, 76].

If a 3-valued partial model P only contains 1 and 0 values, and there is no $\sim$ relation between different objects (i.e. all equivalent nodes are merged), then P also represents a *concrete instance model* M.

Example 1. Figure 1 shows a metamodel extracted from Yakindu statecharts where `Regions` contain `Vertexes` and `Transitions` (leading from a `source` vertex to a `target` vertex). An abstract state `Vertex` is further refined into `States` and `Entry` states where `States` are further refined into `Regions`.

Figure 2 illustrates two partial models: P_4, P_{12} (to be derived by the refinement approach in Section 4). The truth value of the type predicates are denoted by labels on the nodes, where 0 values are omitted. Reference predicate values 1 and $1/2$ are visually represented by edges with solid and dashed lines, respectively, while missing edges between two objects represent 0 values for a predicate. Finally, uncertain $1/2$ equivalences are marked by dashed lines with an $\sim$ symbol, while certain equivalence self-loops on objects are omitted.

Partial model P_4 contains one (concrete) `Region` r, one `State` s, and some other objects collectively represented by a single node $new3$. Object s is both of type `State` and `Vertex`, while $new3$ represents objects with multiple possible types. Object s is linked from r via a `vertices` edge, and there are other possible references between r and $new3$. Partial model P_{12}, which is a refinement of P_4, has no uncertain elements, thus it is also a concrete instance model M.

3.3 Graph patterns as well-formedness constraints

In many industrial modeling tools, complex structural WF constraints are captured either by OCL constraints [70] or by graph patterns (GP) [11,49,67]. Here, we use a tool-independent first-order graph logic representation (which was in-

$$\llbracket \texttt{C}(v) \rrbracket_Z^P := \mathcal{I}_P(\texttt{C})(Z(v))$$
$$\llbracket \texttt{R}(v_1, v_2) \rrbracket_Z^P := \mathcal{I}_P(\texttt{R})(Z(v_1), Z(v_2))$$
$$\llbracket v_1 \sim v_2 \rrbracket_Z^P := \mathcal{I}_P(\sim)(Z(v_1), Z(v_2))$$
$$\llbracket \varphi_1 \wedge \varphi_2 \rrbracket_Z^P := min(\llbracket \varphi_1 \rrbracket_Z^P, \llbracket \varphi_2 \rrbracket_Z^P)$$
$$\llbracket \varphi_1 \vee \varphi_2 \rrbracket_Z^P := max(\llbracket \varphi_1 \rrbracket_Z^P, \llbracket \varphi_2 \rrbracket_Z^P)$$
$$\llbracket \neg \varphi \rrbracket_Z^P := 1 - \llbracket \varphi \rrbracket_Z^P$$
$$\llbracket \exists v : \varphi \rrbracket_Z^P := max\{\llbracket \varphi \rrbracket_{Z, v \mapsto x}^P : x \in Obj_P\}$$
$$\llbracket \forall v : \varphi \rrbracket_Z^P := min\{\llbracket \varphi \rrbracket_{Z, v \mapsto x}^P : x \in Obj_P\}$$
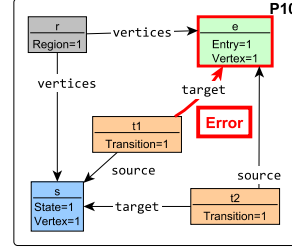

Fig. 3: Semantics of graph patterns (predicates)   Fig. 4: Malformed model

fluenced by [76,98] and is similar to [85]) that covers the key features of several existing graph pattern languages and a first-order logic (FOL) fragment of OCL.

Syntax. A graph pattern (or formula) is a first order logic (FOL) formula $\varphi(v_1, \ldots, v_n)$ over (object) variables. A graph pattern φ can be inductively constructed (see Figure 3) by using atomic predicates of partial models: $\texttt{C}(v)$, $\texttt{R}(v_1, v_2)$, $v_1 \sim v_2$, standard FOL connectives $\neg$, $\vee$, $\wedge$, and quantifiers $\exists$ and $\forall$. A simple graph pattern only contains (a conjunction of) atomic predicates.

Semantics. A graph pattern $\varphi(v_1, \ldots, v_n)$ can be evaluated on partial model P along a variable binding Z, which is a mapping $Z : \{v_1, \ldots, v_n\} \to Obj_P$ from variables to objects in P. The truth value of φ can be evaluated over a partial model P and mapping Z (denoted by $\llbracket \varphi(v_1, \ldots, v_n) \rrbracket_Z^P$) in accordance with the semantic rules defined in Figure 3. Note that min and max takes the numeric minimum and maximum values of 0, $^1/_2$ and 1 with $0 \leq {}^1/_2 \leq 1$, and the rules follow 3-valued interpretation of standard FOL formulae as defined in [76,85].

A variable binding Z is called a *match* if the pattern φ is evaluated to 1 over P, formally $\llbracket \varphi(v_1, \ldots, v_n) \rrbracket_Z^P = 1$. If there exists such a variable binding Z, then we may shortly write $\llbracket \varphi \rrbracket^P = 1$. Open formulae (with one or more unbound variables) are treated by introducing an (implicit) existential quantifier over unbound variables to handle them similarly to graph formulae for regular instance models. Thus, in the sequel, $\llbracket \varphi(v_1, \ldots, v_n) \rrbracket_Z^P = 1$ if $\llbracket \exists v_1, \ldots, \exists v_n : \varphi(v_1, \ldots, v_n) \rrbracket^P = 1$ where the latter is now a closed formula without unbound variables. Similarly, $\llbracket \varphi \rrbracket^P = {}^1/_2$ means that there is a potential match where φ evaluates to $^1/_2$, i.e. $\llbracket \exists v_1, \ldots, \exists v_n : \varphi(v_1, \ldots, v_n) \rrbracket^P = {}^1/_2$, but there is no match with $\llbracket \varphi(v_1, \ldots, v_n) \rrbracket_Z^P = 1$. Finally, $\llbracket \varphi \rrbracket^P = 0$ means that there is surely no match, i.e. $\llbracket \exists v_1, \ldots, \exists v_n : \varphi(v_1, \ldots, v_n) \rrbracket^P = 0$ for all variable bindings. Here $\exists v_1, \ldots, \exists v_n : \varphi(v_1, \ldots, v_n)$ abbreviates $\exists v_1 : (\ldots, \exists v_n : \varphi(v_1, \ldots, v_n))$.

The formal semantics of graph patterns defined in Figure 3 can also be evaluated on regular instance models with closed world assumption. Moreover, if a partial model is also a concrete instance model, the 3-valued and 2-valued truth evaluation of a graph pattern is unsurprisingly the same, as shown in [85].

Proposition 1. *Let P be a partial model which is simultaneously an instance model, i.e. $P = M$. Then the 3-valued evaluation of any φ on P and its 2-valued evaluation on M is identical, i.e. $[\![\varphi]\!]_Z^P = [\![\varphi]\!]_Z^M$ along any variable binding Z.*

Graph patterns as WF constraints. Graph patterns are frequently used for defining complex structural WF constraints and validation rules [96]. Those constraints are derived from two sources: the metamodel (or type graph) defines core structural constraints, and additional constraints of a domain can be defined by using nested graph conditions [40], OCL [70] or graph pattern languages [96].

When specifying a WF constraint ϕ by a graph pattern φ, pattern φ captures the malformed case by negating ϕ, i.e. $\varphi = \neg\phi$. Thus *a graph pattern match detects a constraint violation.* Given a set of graph patterns $\{\varphi_1, \ldots, \varphi_n\}$ constructed that way, a consistent instance model requires that no graph pattern φ_i has a match in M. Thus any match Z for any pattern φ_i with $[\![\varphi_i]\!]_Z^M = 1$ is a proof of inconsistency. In accordance with the consistency definition $M \models WF$ of Section 2, *WF* can defined by graph patterns as $WF = \neg\varphi_1 \wedge \ldots \wedge \neg\varphi_n$.

Note that consistency is defined above only for instance models, but not for partial models. The refinement calculus to be introduced in Section 4 ensures that, by evaluating those graph patterns over partial models, the model generation will gradually converge towards a consistent instance model.

Example 2. The violating cases of two WF constraints checked by the Yakindu tool can be captured by graph patterns as follows:

- $incomingToEntry(v) : \texttt{Entry}(v) \wedge \exists t : \texttt{target}(t, v)$
- $noEntryInRegion(r) : \texttt{Region}(r) \wedge \forall v : \neg(\texttt{vertices}(r, v) \wedge \texttt{Entry}(v))$

Both constraints are satisfied in instance model P_{12} as the corresponding graph patterns have no matches, thus P_{12} is a consistent result of model generation. On the other hand, P_{10} in Figure 4 is a malformed instance model that violates constraint $incomingToEntry(v)$ along object e:

$$[\![incomingToEntry(v)]\!]_{v \mapsto e}^{P_{10}} = 1 \text{ and } [\![noEntryInRegion(r)]\!]^{P_{10}} = 0$$

While graph patterns can be easily evaluated on concrete instance models, checking them over a partial model is a challenging task, because one partial model may represent multiple concretizations. It is shown in [85] how a graph pattern φ can be evaluated on a partial model $\mathcal{P}$ with 3- valued logic and open-world semantics using a regular graph query engine by proposing a constraint rewriting technique. Alternatively, a SAT-solver based approach can be used as in [24, 31] or the general or initial satisfaction can be defined for positive nested graph constraints as in [41, 82].

4 Refinement and Concretization of Partial Models

Model generation is intended to be carried out by a sequence of refinement steps which starts from a generic initial partial model and gradually derives a

concrete instance model. Since our focus is to derive consistent models, we aim at continuously ensuring that each intermediate partial model can potentially be refined into a consistent model, thus a partial model should be immediately excluded if it cannot be extended to a well-formed instance model.

4.1 A refinement relation for partial model generation

In our model generation, the level of uncertainty is aimed to be reduced step by step along a refinement relation which results in partial models that represent a fewer number of concrete instance models than before. In a refinement step, predicates with $1/2$ values can be refined to either 0 or 1, but predicates already fixed to 1 or 0 cannot be changed any more. This imposes an information ordering relation $X \sqsubseteq Y$ where either $X = 1/2$ and Y takes a more specific 1 or 0, or values of X and Y remain equal: $X \sqsubseteq Y := (X = 1/2) \vee (X = Y)$.

Refinement from partial model P to Q (denoted by $P \sqsubseteq Q$) is defined as a function $refine : Obj_P \rightarrow 2^{Obj_Q}$ which maps each object of a partial model P to a non-empty set of objects in the refined partial model Q. Refinement respects the information ordering of type, reference and equivalence predicates for each $p_1, p_2 \in Obj_P$ and any $q_1, q_2 \in Obj_Q$ with $q_1 \in refine(p_1)$, $q_2 \in refine(p_2)$:

- for each class $\texttt{C}_i$: $[\![\texttt{C}_i(p_1)]\!]^P \sqsubseteq [\![\texttt{C}_i(q_1)]\!]^Q$;
- for each reference $\texttt{R}_j$: $[\![\texttt{R}_j(p_1, p_2)]\!]^P \sqsubseteq [\![\texttt{R}_j(q_1, q_2)]\!]^Q$;
- $[\![p_1 \sim p_2]\!]^P \sqsubseteq [\![q_1 \sim q_2]\!]^Q$.

At any stage during refinement, a partial model P can be concretized into an instance model M by rewriting all class type and reference predicates of value $1/2$ to either 1 or 0, and setting all equivalence predicates with $1/2$ to 0 between different objects, and to 1 on a single object. But any concrete instance model will still remain a partial model as well.

Example 3. Figure 5 depicts two sequences of partial model refinement steps deriving two instance models P_{10} (identical to Figure 4) and P_{12} (bottom of Figure 2): $P_0 \sqsubseteq P_4 \sqsubseteq P_5 \sqsubseteq P_6 \sqsubseteq P_7 \sqsubseteq P_8 \sqsubseteq P_9 \sqsubseteq P_{10}$ and $P_0 \sqsubseteq P_4 \sqsubseteq P_5 \sqsubseteq P_6 \sqsubseteq P_7 \sqsubseteq P_8 \sqsubseteq P_{11} \sqsubseteq P_{12}$.

Taking refinement step $P_4 \sqsubseteq P_5$ as an illustration, object $new3$ (in P_4) is refined into e and $new4$ (in P_5) where $[\![e \sim new4]\!]^{P_5} = 0$ to represent two different objects in the concrete instance models. Moreover, all incoming and outgoing edges of $new3$ are copied in e and $new4$. The final refinement step $P_{11} \sqsubseteq P_{12}$ concretizes uncertain `source` and `target` references into concrete references.

A model generation process can be initiated from an initial partial model provided by the user, or from the *most generic partial model* P_0 from which all possible instance models can be derived via refinement. Informally, this P_0 is more abstract than regular metamodels or type graphs as it only contains a single node as top-level class. P_0 contains one abstract object where all predicates are undefined, i.e. $P_0 = \langle Obj_{P_0}, \mathcal{I}_{P_0} \rangle$ where $Obj_{P_0} = \{new\}$ and $\mathcal{I}_{P_0}$ is defined as:
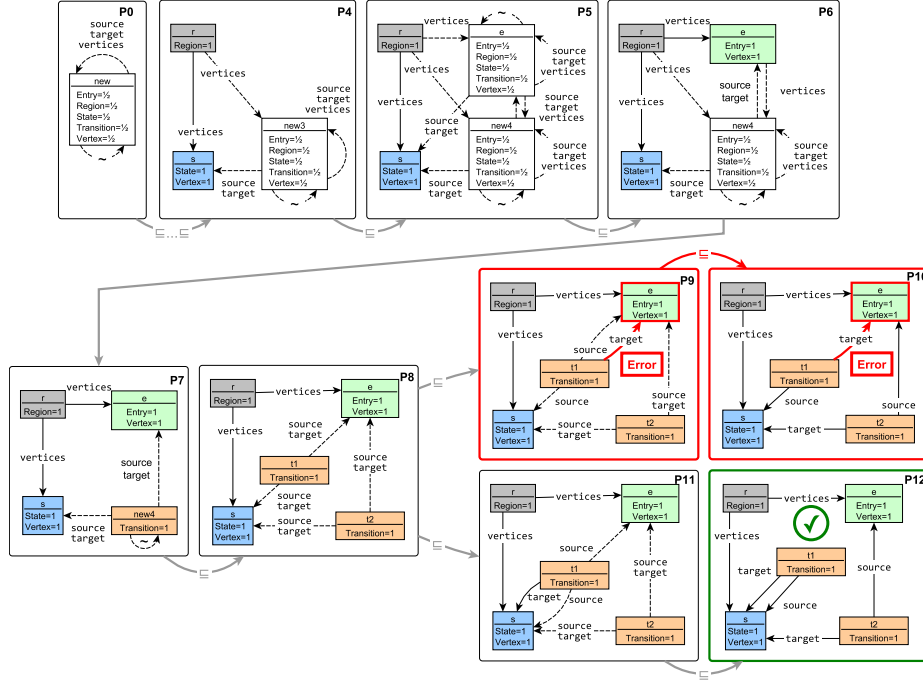
Fig. 5: Refinement of partial models

1. for all class predicates C_i: $[\![C_i(new)]\!]^{P_0} = 1/2$;
2. for all reference predicates R_j: $[\![R_j(new, new)]\!]^{P_0} = 1/2$.
3. $[\![new \sim new]\!]^{P_0} = 1/2$ to represent multiple objects of any instance model;

Our refinement relation ensures that if a predicate is evaluated to either 1 or 0 then its value will no longer change during further refinements as captured by the following approximation theorem.

Theorem 1. *Let P, Q be partial models with $P \sqsubseteq Q$ and φ be a graph pattern.*

- *If $[\![\varphi]\!]^P = 1$ then $[\![\varphi]\!]^Q = 1$; if $[\![\varphi]\!]^P = 0$ then $[\![\varphi]\!]^Q = 0$ (called **under-approximation**).*
- *If $[\![\varphi]\!]^Q = 0$ then $[\![\varphi]\!]^P \leq 1/2$; if $[\![\varphi]\!]^Q = 1$ then $[\![\varphi]\!]^P \geq 1/2$ (called **over-approximation**).*

If model generation is started from P_0 where all (atomic) graph patterns evaluate to $1/2$, this theorem ensures that if a WF constraint φ is violated in a partial model P then it can never be completed to a consistent instance model. Thus the model generation can terminate along this path and a new refinement can be explored after backtracking. This theorem also ensures that if we evaluate a constraint φ on a partial model P and on its refinement Q, the latter will be more precise. In other terms, if $[\![\varphi]\!]^P = 1$ (or 0) in a partial model P along

some sequence of refinement steps, then under-approximation ensures that its evaluation will never change again along that (forward) refinement sequence, i.e. $\llbracket \varphi \rrbracket^Q = 1$ (or 0). Similarly, when proceeding backwards in a refinement chain, over-approximation ensures monotonicity of the 3-valued constraint evaluation along the entire chain. Altogether, we gradually converge to the 2-valued truth evaluation of the constraint on an instance model where less and less constraints take the $1/2$ value. However, a refinement step does not guarantee in itself that exploration is progressing towards a consistent model, i.e. there may be infinite chains of refinement steps which never derive a concrete instance model.

4.2   Refinement operations for partial models

We define *refinement operations Op* to refine partial models by simultaneously growing the size of the models while reducing uncertainty in a way that each finite and consistent instance model is guaranteed to be derived in finite steps.

- *concretize*(p, val): if the atomic predicate p (which is either $\mathtt{C}_i(o)$, $\mathtt{R}_j(o_k, o_l)$ or $o_k \sim o_l$) has a $1/2$ value in the pre-state partial model P, then it can be refined in the post-state Q to val which is either a 1 or 0 value. As an effect of the rule, the level of uncertainty will be reduced.
- *splitAndConnect*$(o, mode)$: if o is an object with $\llbracket o \sim o \rrbracket^P = 1/2$ in the pre-state, then a new object new is introduced in the post state by splitting o in accordance with the semantics defined by the following two modes:
    - *at-least-two*: $\llbracket new \sim new \rrbracket^Q = 1/2$, $\llbracket o \sim o \rrbracket^Q = 1/2$, $\llbracket new \sim o \rrbracket^Q = 0$;
    - *at-most-two*: $\llbracket new \sim new \rrbracket^Q = 1$, $\llbracket o \sim o \rrbracket^Q = 1$, $\llbracket new \sim o \rrbracket^Q = 1/2$;

  In each case, $Obj\,Q = Obj\,P \cup \{new\}$, and we copy all incoming and outgoing binary relations of o to new in Q by keeping their original values in P. Furthermore, all class predicates remain unaltered.

On the technical level, these refinement operations could be easily captured by means of algebraic graph transformation rules [28] over typed graphs. However, for efficiency reasons, several elementary operations may need to be combined into compound rules. Thererfore, specifying refinement operations by graph transformation rules will be investigated in a future paper.

Example 4. Refinement $P_4 \sqsubseteq P_5$ (in Figure 5) is a result of applying refinement operation *splitAndConnect*$(o, mode)$ on object $new3$ and in *at-least-two* mode, splitting $new3$ to e and $new4$ copying all incoming and outgoing references. Next, in P_6, the type of object e is refined to $\mathtt{Entry}$ and $\mathtt{Vertex}$, the $1/2$ equivalence is refined to 1, and references incompatible with $\mathtt{Entry}$ or $\mathtt{Vertex}$ are refined to 0. Note that in P_6 it is ensured that $\mathtt{Region}$ r has an $\mathtt{Entry}$, thus satisfying WF constraint *noEntryInRegion*. In P_7 the type of object $new4$ is refined to $\mathtt{Transition}$, the incompatible references are removed similarly, but the $1/2$ self equivalence remain unchanged. Therefore, in P_8 object $new4$ can split into two separate $\mathtt{Transition}$s: $t1$ and $t2$ with the same source and target options. Refinement $P_8 \sqsubseteq P_9 \sqsubseteq P_{10}$ denotes a possible refinement path, where the $\mathtt{target}$ of

$t1$ is directed to an `Entry`, thus violating WF constraint *incomingToEntry*. Note that this violation can be detected earlier in an unfinished partial model P_9. Refinement $P_{11} \sqsubseteq P_{12}$ denotes the consecutive application of six *concretize*(p, val) operations on uncertain `source` and `target` edges leading out of $t1$ and $t2$ in P_{11}, resulting in a valid model.

Note that these refinement operations may result in a partial model that is unsatisfiable. For instance, if all class predicates evaluate to 0 for an object o of the partial model P, i.e. $[\![C(o)]\!]^P = 0$, then no instance models will correspond to it as most metamodeling techniques require that each element has exactly or at least one type. Similarly, if we violate the reflexivity of $\sim$, i.e. $[\![o \sim o]\!]^P = 0$, then the partial model cannot be concretized into a valid instance model. But at least, one can show that these refinement operations ensure a refinement relation between the partial models of its pre-state and post-state.

Theorem 2 (Refinement operations ensure refinement).
Let P be a partial model and op be a refinement operation. If Q is the partial model obtained by executing op on P (formally, $P \xrightarrow{op} Q$) then $P \sqsubseteq Q$.

4.3   Consistency of model generation by refinement operations

Next we formulate and prove the consistency of model generation when it is carried out by a sequence of refinement steps from the most generic partial model P_0 using the previous refinement operations. We aim to show soundness (i.e. if a model is derivable along an open derivation sequence then it is consistent), finite completeness (i.e. each finite consistent model can be derived along some open derivation sequence), and a concept of incrementality.

Many tableaux based theorem provers build on the concept of closed branches with a contradictory set of formulae. We adapt an analogous concept for closed derivation sequences over graph derivations in [28]. Informally, refinement is not worth being continued as a WF constraint is surely violated due to a match of a graph pattern in case of a closed derivation sequence. Consequently, all consistent instance models will be derived along open derivation sequences.

Definition 1 (Closed vs. open derivation sequence). *A finite derivation sequence of refinement operations $op_1; \ldots; op_k$ leading from the most generic partial model P_0 to the partial model P_k (denoted as $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$) is closed wrt. a graph predicate φ if φ has a match in P_k, formally, $[\![\varphi]\!]^{P_k} = 1$.*
A derivation sequence is open if it is not closed, i.e. P_k is a partial model derived by a finite derivation sequence $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$ with $[\![\varphi]\!]^{P_k} \leq 1/2$.

Note that a single match of φ makes a derivation sequence to be closed, while an open derivation sequence requires that $[\![\varphi]\!]^{P_k} \leq 1/2$ which, by definition, disallows a match with $[\![\varphi]\!]^{P_k} \leq 1$.

Example 5. Derivation sequence $P_0 \dashrightarrow P_9$ depicted in Figure 5 is closed for $\varphi = incomingToEntry(v)$ as the corresponding graph pattern has a match in P_9,

i.e. $[\![incomingToEntry(v)]\!]_{v \mapsto e}^{P_9} = 1$. Therefore, P_{10} can be avoided as the same match would still exist. On the other hand, derivation sequence $P_0 \dashrightarrow P_{11}$ is open for $\varphi = incomingToEntry(v)$ as $incomingToEntry(v)$ is evaluated to $1/2$ in all partial models $P_0, \ldots, P_{11}$.

As a consequence of Theorem 1, an open derivation sequence ensures that any prefix of the same derivation sequence is also open.

Corollary 1. *Let $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$ be an open derivation sequence of refinement operations wrt. φ. Then for each $0 \leq i \leq k$, $[\![\varphi]\!]^{P_i} \leq 1/2$.*

The soundness of model generation informally states that if a concrete model M is derived along an open derivation sequence then M is consistent, i.e. no graph predicate of WF constraints has a match.

Corollary 2 (Soundness of model generation). *Let $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$ be a finite and open derivation sequence of refinement operations wrt. φ. If P_k is a concrete instance model M (i.e. $P_k = M$) then M is consistent (i.e. $[\![\varphi]\!]^M = 0$).*

Effectively, once a concrete instance model M is reached during model generation along an open derivation sequence, checking the WF constraints on M by using traditional (2-valued) graph pattern matching techniques ensures the soundness of model generation as 3-valued and 2-valued evaluation of the same graph pattern should coincide due to Proposition 1 and Theorem 1.

Next, we show that any finite instance model can be derived by a finite derivation sequence.

Theorem 3 (Finiteness of model generation). *For any finite instance model M, there exists a finite derivation sequence $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$ of refinement operations starting from the most generic partial model P_0 leading to $P_k = M$.*

Our completeness theorem states that any consistent instance model is derivable along open derivation sequences where no constraints are violated (under-approximation). Thus it allows to eliminate all derivation sequences where an graph predicate φ evaluates to 1 on any intermediate partial model P_i as such partial model cannot be further refined to a well-formed concrete instance model due to the properties of under-approximation. Moreover, a derivation sequence leading to a consistent model needs to be open wrt. all constraints, i.e. refinement can be terminated if any graph pattern has a match.

Theorem 4 (Completeness of model generation). *For any finite and consistent instance model M with $[\![\varphi]\!]^M = 0$, there exists a finite open derivation sequence $P_0 \xrightarrow{op_1;\ldots;op_k} P_k$ of refinement operations wrt. φ starting from the most generic partial model P_0 and leading to $P_k = M$.*

Unsurprisingly, graph model generation still remains undecidable in general as there is no guarantee that a derivation sequence leading to P_k where $[\![\varphi]\!]^{P_k} = 1/2$ can be refined later to a consistent instance model M. However, the graph

model finding problem is decidable for a finite scope, which is an a priori upper bound on the size of the model. Informally, since the size of partial models is gradually growing during refinement, we can stop if the size of a partial model exceeds the target scope or if a constraint is already violated.

Theorem 5 (Decidability of model generation in finite scope). *Given a graph predicate φ and a scope $n \in \mathbb{N}$, it is decidable to check if a concrete instance model M exists with $|Obj_M| \leq n$ where $[\![\varphi]\!]^M = 0$.*

This finite decidability theorem is analogous with formal guarantees provided by the Alloy Analyzer [94] that is used by many mapping-based model generation approaches (see Section 6). Alloy aims to synthesize small counterexamples for a relational specification, while our refinement calculus provides the same for typed graphs without parallel edges for the given refinement operations.

However, our construction has extra benefits compared to Alloy (and other SAT-solver based techniques) when exceeding the target scope. First, all candidate partial models (with constraints evaluated to $1/2$) derived up to a certain scope are reusable for finding consistent models of a larger scope, thus search can be incrementally continued. Moreover, if a constraint violation is found with a given scope, then no consistent models exist at all.

Corollary 3 (Incrementality of model generation). *Let us assume that no consistent models M^n exist for scope n, but there exists a larger consistent model M^m of size m (where $m > n$) with $[\![\varphi]\!]^{M^m} = 0$. Then M^m is derivable by a finite derivation sequence $P_i^n \xrightarrow{op_{i+1};...;op_k} P_k^m$ where $P_k^m = M^m$ starting from a partial model P_i^n of size n.*

Corollary 4 (Completeness of refutation). *If all derivation sequences are closed for a given scope n, but no consistent model M^n exists for scope n for which $[\![\varphi]\!]^{M^n} = 0$, then no consistent models exist at all.*

While these theorems aim to establish the theoretical foundations of a model generator framework, it provides no direct practical insight on the exploration itself, i.e. how to efficiently provide derivation sequences that likely lead to consistent models. Nevertheless, we have an initial prototype implementation of such a model generator which is also used as part of the experimental evaluation.

5 Evaluation

As existing model generators have been dominantly focusing on a single challenge of Section 2, we carried out an initial experimental evaluation to investigate how popular strategies excelling in one challenge perform with respect to another challenge. More specifically, we carried out this evaluation in the domain of Yakindu statecharts to address four research questions:

RQ1 *Diverse vs. Realistic:* How realistic are the models which are derived by random generators that promise diversity?

RQ2 *Consistent vs. Realistic:* How realistic are the models which are derived by logic solvers that guarantee consistency?

RQ3 *Diverse vs. Consistent:* How consistent are the models which are derived by random generators?

RQ4 *Consistent vs. Scalable:* How scalable is it to evaluate consistency constraints on partial models?

Addressing these questions may help advancing future model generators by identifying some strength and weaknesses of different strategies.

5.1 Setup of experiments

Target domain We conducted measurements in the context of Yakindu statecharts, see [2] for the complete measurement data. For that purpose, we extracted the statechart metamodel of Figure 1 directly from the original Yakindu metamodel. Ten WF constraints were formalized as graph patterns based on the real validation rules of the Yakindu Statechart development environment.

Model generator approaches For addressing *RQ1-3*, we used two different model generation approaches: (1) the popular *relational model finder* Alloy Analyzer [94] which uses Sat4j [53] as a back-end SAT-solver, and (2) the VIATRA Solver, *graph-based model generator* which uses the refinement calculus presented in the paper. We selected Alloy Analyzer as the primary target platform as it has been widely used in mapping based generators of consistent models (see Section 6).

We operated these solvers in two modes: in *consistent* mode (WF), all derived models need to satisfy all WF constraints of Yakindu statecharts, while in *metamodel-only* mode (MM), generated models need to be metamodel compliant, but then model elements are selected randomly. As such, we expect that this set of models is diverse, but the fulfillment of WF constraints is not guaranteed. To enforce diversity, we explicitly check that derived models are non-isomorphic.

Since mapping based approaches typically compile WF constraints into logic formulae in order to evaluate them on partial models, we set up a simple measurement to address *RQ4* which did not involve model generation but only constraint checking on existing instance models. This is a well-known setup for assessing scalability of graph query techniques used in a series of benchmarking papers [92, 98]. So in our case, we encoded instance models as fully defined Alloy specifications using the mapping of [86], and checked if the constraints are satisfied (without extending or modifying the statechart). As a baseline of comparison, we checked the runtime of evaluating the same WF constraints on the same models using an industrial graph query engine [97] which is known to scale well for validation problems [92, 98]. All measurements were executed on an average desktop computer[6].

---

[6] CPU: Intel Core-i5-m310M, MEM: 16GB, OS: Windows 10 Pro.

Real instance models To evaluate how realistic the synthetic model generators are in case of *RQ1-2*, we took 1253 statecharts as *real models* created by undergraduate students for a homework assignment. While they had to solve the same modeling problem, the size of their models varied from 50 to 200 objects. For *RQ4*, we randomly selected 300 statecharts from the homework assignments, and evaluated the original validation constraints. Real models were filtered by removing inverse edges that introduce significant noise to the metrics [91].

Generated instance models To obtain comparable results, we generated four sets of statechart models with a timeout of 1 minute for each model but without any manual domain-specific fine-tuning of the different solvers. We also check that the generated models are non-isomorphic to assure sufficient diversity.

- Alloy (MM): 100 metamodel-compliant models with 50 objects using Alloy.
- Alloy (WF): 100 metamodel- and WF-compliant models with 50 objects using Alloy (which was unable to synthesize larger models within 1 minute).
- VIATRA Solver (MM): 100 metamodel-compliant instance models with 100 objects using VIATRA Solver.
- VIATRA Solver (WF): 100 Metamodel- and WF-compliant instance models with 100 objects using VIATRA Solver.

Two multi-dimensional graph metrics are used to evaluate how realistic a model generator is: (1) the *multiplex participation coefficient (MPC)* measures how the edges of nodes are distributed along the different edge types, while (2) *pairwise multiplexity (Q)* captures how often two different types of edges meet in an object. These metrics were recommended in [91] out of over 20 different graph metrics after a multi-domain experimental evaluation, for formal definitions of the metrics, see [91]. Moreover, we calculate the (3) number of WF constraints violated by a model as a numeric value to measure the degree of (in)consistency of a model (which value is zero in case of consistent models).

5.2 Evaluation of measurement results

We plot the distribution functions of the *multiple participation coefficient* metric in Figure 6, and the *pairwise multiplexity* metric in Figure 7. Each line depicts the characteristics of a single model and model sets (e.g. "Alloy (MM)", "VIATRA Solver (WF)") are grouped together in one of the facets including the characteristics of the real model set. For instance, the former metric tells that approximately 65% of nodes in real statechart models (right facet in Figure 6) have only one or two types of incoming and outgoing edges while the remaining 35% of nodes have edges more evenly distributed among different types.

Comparison of distribution functions. We use visual data analytics techniques and the Kolmogorov-Smirnov statistic (*KS*) [55] as a distance measure of models (used in [91]) to judge how realistic an auto-generated model is by comparing the whole distributions of values (and not only their descriptive summary like
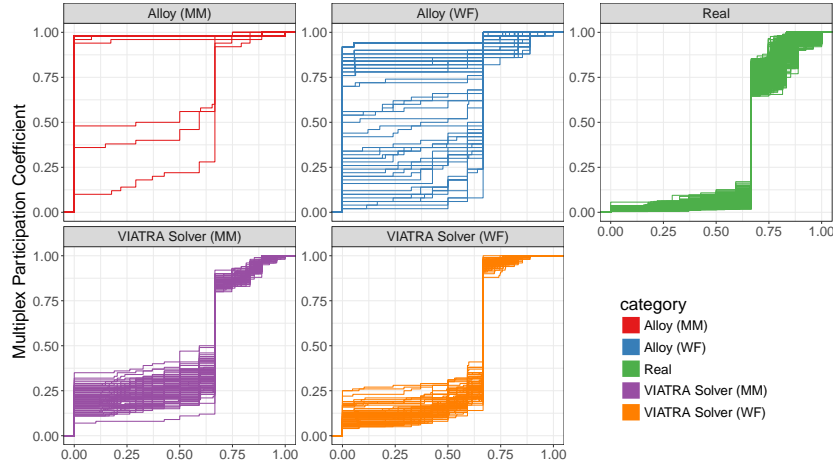
Fig. 6: Measurement results: Multiplex participation coefficient (MPC)

mean or variance) in different cases to the characteristics of real models. The KS statistics quantifies the maximal difference between the distribution function lines at a given value. It is sensitive to both shape and location differences: it takes a 0 value only if the distributions are identical, while it is 1 if the values of models are in disjunct ranges (even if their shapes are identical). For comparing model generation techniques A and B we took the average value of the KS statistics between each (A, B) pair of models that were generated by technique A and B, respectively. The average KS values are shown in Figure 9,[7] where a lower value denotes a more realistic model set.

Diverse vs. Realistic: For the models that are only metamodel-compliant, the characteristics of the metrics for "VIATRA Solver (MM)" are much closer to the "Real" model set than those of the "Alloy (MM)" model set, for both graph metrics (KS value of 0.27 vs. 0.95 for MPC and 0.38 vs. 0.88 for Q), thus more realistic results were generated in the "VIATRA Solver (MM)" case. However, these plots also highlight that the set of auto-generated metamodel-compliant models can be more easily distinguished from the set of real models as the plots of the latter show higher variability. Since the diversity of each model generation case is enforced (i.e. non-isomorphic models are dropped), we can draw as a conclusion that *a diverse metamodel-compliant model generator does not provide any guarantees in itself on the realistic nature of the output model set.* In fact, model generators that simultaneously ensure diversity and consistency always outperformed the random model generators for both solvers.

---

[7] Due to the excessive amount of homework models, we took a uniform random sample of 100 models from that model set.
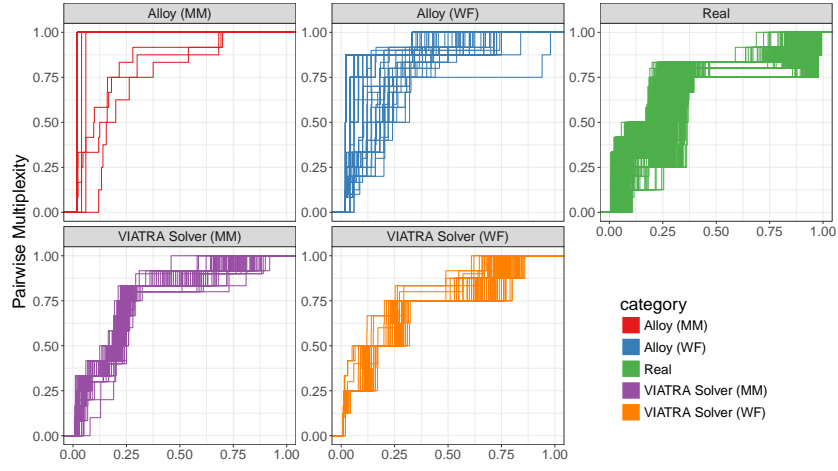
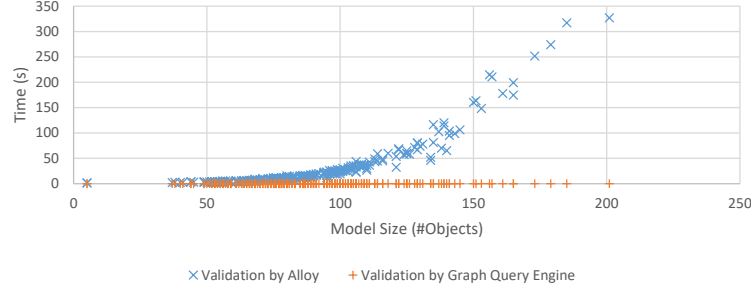Fig. 7: Measurement results: Pairwise multiplexity (Q)

Fig. 8: Measurement results: time of consistency checks: Alloy vs. VIATRA Solver

Consistent vs. Realistic: In case of models satisfying WF constraints "VIATRA Solver (WF)" generated more realistic results than "Alloy (WF)" according to both metrics. The plots show mixed results for differentiating between generated and realistic models. On the positive side, the shape of the plot of auto-generated models is very close to that of real models in case of the *MPC* metric (Figure 6) – statistically, they have a relatively low average *KS* value of 0.24. However, for the *Q* metric (Figure 7), real

model set	MPC	Q
Alloy (MM)	0.95	0.88
Alloy (WF)	0.74	0.60
VIATRA Solver (MM)	0.27	0.37
VIATRA Solver (WF)	0.24	0.30

Fig. 9: Average Kolmogorov-Smirnov statistics between the *real* and generated model sets.

models are still substantially different from generated ones (average *KS* value of 0.3). Thus *further research is needed to investigate how to make consistent models more realistic.*

Diverse vs. Consistent: We also calculated the average number of WF constraint violations, which was 3.1 for the "Alloy (MM)" case and 9.75 for the "VIATRA Solver (MM)" case, while only 0.07 for real models. We observe that a diverse set of randomly generated metamodel-compliant instance models do not yield consistent models as some constraints will always be violated – which is not the case for real statechart models. In other terms, the number of WF constraint violations is also an important characteristic of realistic models which is often overseen in practice. As a conclusion, a *model generator should ensure consistency prior to focusing on increasing diversity.* Since humans dominantly come up with consistent models, *ensuring consistency for realistic models is a key prerequisite.*

Consistent vs. Scalable: The soundness of consistent model generation inherently requires the evaluation of the WF constraints at least once for a candidate solution. Figure 8 depicts the validation time of randomly selected homework models using Alloy and the VIATRA graph query engine wrt. the size of the instance model (i.e. the number of the objects). For each model, the two validation techniques made the same judgment (as a test for their correctness). Surprisingly, the diagram shows that the Alloy Analyzer scales poorly for evaluating constraints on medium-size graphs, which makes it unsuitable for generating larger models. The runtime of the graph query engine was negligible at this scale as we expected based on detailed previous benchmarks for regular graph pattern matching and initial experiments for matching constraints over partial models [85].

While many existing performance benchmarks claim that they generate realistic models, most of them ignore WF constraints of the domain. According to our measurements, it is a major drawback since real models dominantly satisfy WF constraints while randomly generated models always violate some constraint. This way, those model generators can hardly be considered realistic.

Threats to validity We carried out experiments only in the domain of statecharts which limits the generalizability of our results. Since statecharts are a behavioral modeling language, the characteristics of models (and thus the graph metrics) would likely differ from e.g. static architectural DSLs. However, since many of our experimental results are negative, it is unlikely that the Alloy generator would behave any better for other domains. It is a major finding that while Alloy has been used as a back-end for mapping-based model generator approaches, its use is not justified from a scalability perspective due to the lack of efficient evaluation for complex structural graph constraints. It is also unlikely that randomly generated metamodel-compliant models would be more realistic, or more consistent in any other domains.

Concerning our real models, we included all the statecharts created by students, which may be a bias since students who obtained better grades likely

		Logic Solvers	Random Generators	Network Graphs	Performance Benchmarks	Real Dataset
CON	Model	+	−	−	+	+
	Complete	0	−	−	−	−
DIV	Model	−	+	−	−	−
	Set	−	+	−	−	−
SCA	In Size	−	+	+	+	+
	In Quantity	−	?	+	+	−
REA	Model	-	-	-	-	+
	Set	−	−	−	0	+

Table 1: Characteristics of model generation approaches; +: feature provided, −: feature not provided, 0: feature provided in some tools / cases

produced better quality models. Thus, the variability of real statechart models created by engineers may actually be smaller. But this would actually increase the relative quality of models derived by VIATRA Solver which currently differs from real models by providing a lower level of diversity (i.e. plots of pairwise multiplicity are thicker for real models).

6 Related work

We assess and compare how existing approaches address each challenge (Table 1).

Consistent model generators (CON): Consistent models can be synthesized as a side effect of a verification process when aiming to prove the consistency of a DSL specification. The metamodel and a set of WF constraints are captured in a high-level DSL and logic formulae are generated as input to back-end logic solvers. Approaches differ in the *language used for WF constraints*, OCL [18–20, 23, 35, 50–52, 73, 87, 100], graph constraints [84, 86], Java predicates [14] or custom DSLs like Formula [46], Clafer [8] or Alloy [45]. They also differ in *the solver used in the background*: graph transformation engines as in [100], SAT-solvers [53] are used in [51, 52], model finders like Kodkod [94] are target formalisms in [5, 23, 50, 87], first-order logic formulae are derived for SMT-solvers [65] in [73, 84] while CSP-solvers like [1] are targeted in [18, 19] or other techniques [59, 74].

Solver-based approaches excel in finding inconsistencies in specifications, while the generated model is a proof of consistency. While SAT solvers can handle specifications with millions of Boolean variables, all these mapping-based techniques still suffer from severe scalability issues as the generated graphs may contain less than 50-100 nodes. This is partly due to the fact that a Boolean variable needs to be introduced for each *potential* edge in the generated model, which blows up the complexity. Moreover, the output models are highly similar to each other and lack diversity, thus they cannot directly be applied for testing or benchmarking purposes.

Diverse model generators (DIV): Diverse models play a key role in testing model transformations and code generators. Mutation-based approaches [6, 25, 61] take existing models and make random changes on them by applying mutation rules. A similar random model generator is used for experimentation purposes in [9]. Other automated techniques [15, 29] generate models that only conform to the metamodel. While these techniques scale well for larger models, there is no guarantee whether the mutated models satisfy WF constraints.

There is a wide set of model generation techniques which provide certain promises for test effectiveness. White-box approaches [37, 38, 62, 83] rely on the implementation of the transformation and dominantly use back-end logic solvers, which lack scalability when deriving graph models. Black-box approaches [17, 34, 39, 56, 63] can only exploit the specification of the language or the transformation, so they frequently rely upon contracts or model fragments. As a common theme, these techniques may generate a set of simple models, and while certain diversity can be achieved by using symmetry-breaking predicates, they fail to scale for larger model sizes. In fact, the effective diversity of models is also questionable since corresponding safety standards prescribe much stricter test coverage criteria for software certification and tool qualification than those currently offered by existing model transformation testing approaches.

Based on the logic-based Formula solver, the approach of [47] applies stochastic random sampling of output to achieve a diverse set of generated models by taking exactly one element from each equivalence class defined by graph isomorphism, which can be too restrictive for coverage purposes. Stochastic simulation is proposed for graph transformation systems in [95], where rule application is stochastic (and not the properties of models), but fulfillment of WF constraints can only be assured by a carefully constructed rule set.

Realistic model generators (REA): The igraph library [22] contains a set of randomized graph generators that produce one-dimensional (untyped) graphs that follow a particular distribution (e.g. Erdős-Rényi, Watts-Strogatz). The authors of [64] use Boltzmann samplers [27] to ensure efficient generation of uniform models. GSCALER [102] takes a graph as its input and generates a similar graph with a certain number of vertices and edges.

Scalable model generators (SCA): Several database benchmarks provide scalable graph generators with some degree of well-formedness or realism. The *Berlin SPARQL Benchmark (BSBM)* [13] uses a single dataset that scales in model size (10 million–150 billion tuples), but does not vary in structure. SP^2Bench [80] uses a data set, which is synthetically generated based on the real-world DBLP bibliography. This way, instance models of different sizes reflect the structure and complexity of the original real-world dataset.

The Linked Data Benchmark Council (LDBC) recently developed the Social Network Benchmark [30], which contains a social network generator module [88]. The generator is based on the S3G2 approach [72] that aims to generate non-uniform value distributions and structural correlations. gMark [7] generates

graphs driven by a pre-defined schema that allows users to specify vertex/edge types and degree distributions in the graph, which provides some level of realism.

The *Train Benchmark* [92] uses a domain-specific generator that is able to generate railway networks, scalable in size and satisfying a set of well-formedness constraints. The generator is also able to inject errors to the models during generation (thus intentionally violating the WF property).

Transformations of Partial Models Uncertain models [31] document semantic variation points generically by annotations on a regular instance model. Potential concrete models compliant with an uncertain model can be synthesized by the Alloy Analyzer and its back-end SAT solvers [78, 79], or refined by graph transformation rules [77].

Transformations over partial models [32, 33] analyse possible matches and executions of model transformation rules on partial models by using a SAT solver (MathSAT4) or by automated graph approximation called "lifting", which inspects possible partitions of a finite concrete model, i.e. regular graph transformation rules are lifted, while in this paper, we attempt to introduce model generator rules directly on the level of partial models.

Regular graph transformation rules are used for model generation is carried out in [29, 100] where output models are metamodel compliant, but they do not fulfill extra WF constraints of the domain [29] or (a restricted set of) constraints need to be translated first to rule application conditions [100].

Symbolic Model Generation Certain techniques use abstract (or symbolic) graphs for analysis purposes. A tableau-based reasoning method is proposed for graph properties [3,71,81], which automatically refine solutions based on well-formedness constraints, and handle state space in the form of a resolution tree. As a key difference, our approach refines possible solutions in the form of partial models, while [71,81] resolves the graph constraints to a concrete solution.

7   Conclusion and Future Work

In this paper, we presented the challenge of automated graph model generation where models are consistent, diverse, scalable and realistic at the same time. In an experimental evaluation, we found that traditional model generation techniques which excel in one aspect perform poorly with respect to another aspect. Furthermore, consistent models turn out to be a prerequisite both for the realistic and diverse cases. As the main conceptual contribution of this paper, we presented a refinement calculus based on 3-valued logic evaluation of graph patterns that could drive the automated synthesis of consistent models. We proved soundness and completeness for this refinement approach, which also enables to incrementally generate instance models of a larger scope by reusing partial models traversed in a previous scope. As such, it offers stronger consistency guarantees than the popular Alloy Analyzer used as a back-end solver for many mapping-based model generation approaches.

While an initial version of a model generator operating that way was included in our experimental evaluation, our main ongoing work is to gradually address several model generation challenges at the same time. For instance, model generators which are simultaneously consistent, diverse and realistic could help in the systematic testing of the VIATRA transformation framework [97] or other industrial DSL tools.

Acknowledgements

References

1. Choco, `http://www.emn.fr/z-info/choco-solverp`
2. Complete measurement setup and results of the paper (2017), `https://github.com/FTSRG/publication-pages/wiki/Towards-the-Automated-Generation-of-Consistent,-Diverse,-Scalable,-and-Realistic-Graph-Models/`
3. Al-Sibahi, A.S., Dimovski, A.S., Wasowski, A.: Symbolic execution of high-level transformations. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam, The Netherlands, October 31 - November 1, 2016. pp. 207–220 (2016), `http://dl.acm.org/citation.cfm?id=2997382`
4. Ali, S., Iqbal, M.Z.Z., Arcuri, A., Briand, L.C.: Generating test data from OCL constraints with search techniques. IEEE Trans. Software Eng. 39(10), 1376–1402 (2013)
5. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On challenges of model transformation from UML to Alloy. Softw. Syst. Model. 9(1), 69–86 (2010)
6. Aranega, V., Mottu, J.M., Etien, A., Degueule, T., Baudry, B., Dekeyser, J.L.: Towards an automation of the mutation analysis dedicated to model transformation. Softw. Test., Verif. Reliab. 25(5-7), 653–683 (2015)
7. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: gMark: Schema-driven generation of graphs and queries. IEEE Transactions on Knowledge and Data Engineering 29(4), 856–869 (2017)
8. Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A.: Clafer: unifying class and feature modeling. Softw. Syst. Model. 15(3), 811–845 (2016)
9. Batot, E., Sahraoui, H.: A generic framework for model-set selection for the unification of testing and learning MDE tasks. In: MODELS. pp. 374–384. ACM Press (2016)
10. Battiston, F., Nicosia, V., Latora, V.: Structural measures for multiplex networks. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics 89(3) (2014)
11. Bergmann, G., Ujhelyi, Z., Ráth, I., Varró, D.: A graph query language for EMF models. In: ICMT. LNCS, vol. 6707, pp. 167–182. Springer (2011)
12. Berlingerio, M., et al.: Multidimensional networks: foundations of structural analysis. World Wide Web 16(5-6), 567–593 (2013)

13. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Sem. Web & Inf. Sys. 5(2), 1–24 (2009)
14. Boyapati, C., Khurshid, S., Marinov, D.: Korat: Automated testing based on java predicates. In: International Symposium on Software Testing and Analysis (ISSTA. pp. 123–133. ACM Press (2002)
15. Brottier, E., Fleurey, F., Steel, J., Baudry, B., Le Traon, Y.: Metamodel-based test generation for model transformations: an algorithm and a tool. In: ISSRE. pp. 85–94 (Nov 2006)
16. Bures, T., et al.: Software engineering for smart cyber-physical systems – towards a research agenda. ACM SIGSOFT Software Engineering Notes 40(6), 28–32 (2015)
17. Büttner, F., Egea, M., Cabot, J., Gogolla, M.: Verification of ATL transformations using transformation models and model finders. In: ICFEM. pp. 198–213. LNCS 7635, Springer (2012)
18. Cabot, J., Clarisó, R., Riera, D.: On the verification of UML/OCL class diagrams using constraint programming. J. Syst. Softw. 93, 1–23 (2014)
19. Cabot, J., Teniente, E.: Incremental integrity checking of UML/OCL conceptual schemas. J. Syst. Softw. 82(9), 1459–1478 (2009)
20. Clavel, M., Egea, M., de Dios, M.A.G.: Checking unsatisfiability for OCL constraints. ECEASST 24 (2009)
21. Corradini, A., König, B., Nolte, D.: Specifying graph languages with type graphs. In: Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany. LNCS, vol. 10373, pp. 73–89 (2017)
22. Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal Complex Systems, 1695 (2006), http://igraph.sf.net
23. Cunha, A., Garis, A., Riesco, D.: Translating between alloy specifications and uml class diagrams annotated with ocl. Softw. Syst. Model. pp. 1–21 (2013)
24. Czarnecki, K., Pietroszek, K.: Verifying feature-based model templates against well-formedness OCL constraints. In: Generative Programming and Component Engineering, 5th International Conference, GPCE 2006. pp. 211–220. ACM (2006)
25. Darabos, A., Pataricza, A., Varró, D.: Towards testing the implementation of graph transformations. In: GTVMT. ENTCS, Elsevier (2006)
26. DeWitt, D.J.: The Wisconsin Benchmark: Past, present, and future. In: The Benchmark Handbook, pp. 119–165 (1991)
27. Duchon, P., Flajolet, P., Louchard, G., Schaeffer, G.: Boltzmann samplers for the random generation of combinatorial structures. Comb. Probab. Comp. 13(4-5), 577–625 (2004)
28. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006)
29. Ehrig, K., Küster, J.M., Taentzer, G.: Generating instance models from meta models. Softw. Syst. Model 8(4), 479–500 (2009)
30. Erling, O., et al.: The LDBC Social Network Benchmark: Interactive workload. In: SIGMOD. pp. 619–630 (2015)
31. Famelis, M., Salay, R., Chechik, M.: Partial models: Towards modeling and reasoning with uncertainty. In: ICSE. pp. 573–583. IEEE Press (2012)
32. Famelis, M., Salay, R., Chechik, M.: The semantics of partial model transformations. In: MiSE at ICSE. pp. 64–69. IEEE Press (2012)
33. Famelis, M., Salay, R., Di Sandro, A., Chechik, M.: Transformation of models containing uncertainty. In: MODELS. pp. 673–689. Springer (2013)

34. Fleurey, F., Baudry, B., Muller, P.A., Le Traon, Y.: Towards dependable model transformations: Qualifying input test data. Softw. Syst. Model. (2007)
35. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Sci. Comput. Program. 69(1-3), 27–34 (Dec 2007)
36. Goldberg, A.P., Chew, Y.H., Karr, J.R.: Toward scalable whole-cell modeling of human cells. In: SIGSIM-PADS. pp. 259–262. ACM Press (2016)
37. González, C.A., Cabot, J.: ATLTest: A white-box test generation approach for ATL transformations. In: MODELS. LNCS, vol. 7590, pp. 449–464. Springer (2012)
38. Gonzalez, C.A., Cabot, J.: Test data generation for model transformations combining partition and constraint analysis. In: ICMT. pp. 25–41. Springer (2014)
39. Guerra, E., Soeken, M.: Specification-driven model transformation testing. Softw. Syst. Model. 14(2), 623–644 (may 2015)
40. Habel, A., Pennemann, K.: 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. LNCS, vol. 3393, pp. 293–308. Springer (2005)
41. Habel, A., Pennemann, K.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19(2), 245–296 (2009)
42. Härtel, J., Härtel, L., Lämmel, R.: Test-data generation for Xtext. In: SLE. pp. 342–351 (2014)
43. ISO: Road vehicles – Functional safety (ISO 26262) (2011)
44. Izsó, B., Szatmári, Z., Bergmann, G., Horváth, Á., Ráth, I.: Towards precise metrics for predicting graph query performance. In: ASE. pp. 421–431 (2013)
45. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. 11(2), 256–290 (2002)
46. Jackson, E.K., Levendovszky, T., Balasubramanian, D.: Automatically reasoning about metamodeling. Softw. Syst. Model. 14(1), 271–285 (2015)
47. Jackson, E.K., Simko, G., Sztipanovits, J.: Diversely enumerating system-level architectures. In: EMSOFT. p. 11. IEEE Press (2013)
48. Kleene, S.C., De Bruijn, N., de Groot, J., Zaanen, A.C.: Introduction to meta-mathematics, vol. 483. van Nostrand New York (1952)
49. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: On the evolution of ocl for capturing structural constraints in modelling languages. In: Rigorous Methods for Software Construction and Analysis, pp. 204–218 (2009)
50. Kuhlmann, M., Gogolla, M.: From UML and OCL to Relational Logic and Back, LNCS, vol. 7590. Springer (2012)
51. Kuhlmann, M., Gogolla, M.: Strengthening SAT-based validation of UML/OCL models by representing collections as relations. In: ECMFA. LNCS, vol. 7349, pp. 32–48 (2012)
52. Kuhlmann, M., Hamann, L., Gogolla, M.: Extensive validation of OCL models by integrating SAT solving into USE. In: TOOLS - Objects, Models, Components and Patterns. LNCS, vol. 6705, pp. 290–306 (2011)
53. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation 7, 59–64 (2010)
54. Lee, E.A., et al.: The swarm at the edge of the cloud. IEEE Design & Test 31(3), 8–20 (2014)
55. Lehmann, E.L., D'Abrera, H.J.: Nonparametrics: statistical methods based on ranks. Springer NY (2006)

56. López-Fernández, J.J., Guerra, E., de Lara, J.: Combining unit and specification-based testing for meta-model validation and verification. Inf. Syst. 62, 104–135 (2016)
57. Meedeniya, I., Aleti, A., Grunske, L.: Architecture-driven reliability optimization with uncertain model parameters. J. Syst. Softw. 85(10), 2340–2355 (2012)
58. Micskei, Z., Szatmári, Z., Oláh, J., Majzik, I.: A concept for testing robustness and safety of the context-aware behaviour of autonomous systems. In: KES-AMSTA. LNCS, vol. 7327, pp. 504–513. Springer (2012)
59. Misailovic, S., Milicevic, A., Petrovic, N., Khurshid, S., Marinov, D.: Parallel test generation and execution with korat. In: ESEC-FSE '07. pp. 135–144. ACM (2007)
60. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL benchmark: Performance assessment with real queries on real data. In: ISWC. pp. 454–469. Springer (2011)
61. Mottu, J., Baudry, B., Traon, Y.L.: Mutation analysis testing for model transformations. In: ECMDA-FA. LNCS, vol. 4066, pp. 376–390. Springer (2006)
62. Mottu, J.M., Sen, S., Tisi, M., Cabot, J.: Static analysis of model transformations for effective test generation. In: ISSRE. pp. 291–300. IEEE (nov 2012)
63. Mottu, J.M., Simula, S.S., Cadavid, J., Baudry, B.: Discovering model transformation pre-conditions using automatically generated test models. In: ISSRE. pp. 88–99. IEEE (nov 2015)
64. Mougenot, A., Darrasse, A., Blanc, X., Soria, M.: Uniform random generation of huge metamodel instances. In: ECMDA-FA. pp. 130–145 (2009)
65. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
66. Neema, S., Sztipanovits, J., Karsai, G., Butts, K.: Constraint-based design-space exploration and model synthesis. In: EMSOFT. pp. 290–305 (2003)
67. Nickel, U., Niere, J., Zündorf, A.: The FUJABA environment. In: ICSE. pp. 742–745. ACM (2000)
68. Nicosia, V., Latora, V.: Measuring and modeling correlations in multiplex networks. Phys. Rev. E 92, 032805 (2015)
69. Nielsen, C.B., Larsen, P.G., Fitzgerald, J.S., Woodcock, J., Peleska, J.: Systems of systems engineering: Basic concepts, model-based techniques, and research directions. ACM Comput. Surv. 48(2), 18 (2015)
70. The Object Management Group: Object Constraint Language, v2.0 (May 2006)
71. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: International Conference on Graph Transformation. pp. 289–304. Springer (2008)
72. Pham, M., Boncz, P.A., Erling, O.: S3G2: A scalable structure-correlated social graph generator. In: TPCTC. pp. 156–172 (2012)
73. Przigoda, N., Hilken, F., Peters, J., Wille, R., Gogolla, M., Drechsler, R.: Integrating an SMT-based modelfinder into USE. In: Model-Driven Engineering, Verification and Validation (MoDeVVa) at MODELS. vol. 1713, pp. 40–45 (2016)
74. Queralt, A., Artale, A., Calvanese, D., Teniente, E.: OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. Data Knowl. Eng. 73, 1–22 (2012)
75. Rensink, A., Distefano, D.: Abstract graph transformation. Electr. Notes in Theoret. Comp. Sci. 157(1), 39–59 (2006)
76. Reps, T.W., Sagiv, M., Wilhelm, R.: Static program analysis via 3-valued logic. In: CAV. pp. 15–30 (2004)
77. Salay, R., Chechik, M., Famelis, M., Gorzny, J.: A methodology for verifying refinements of partial models. Journal of Object Technology 14(3), 3:1–31 (2015)

78. Salay, R., Chechik, M., Gorzny, J.: Towards a methodology for verifying partial model refinements. In: ICST. pp. 938–945. IEEE (2012)
79. Salay, R., Famelis, M., Chechik, M.: Language independent refinement using partial modeling. In: FASE, pp. 224–239. LNCS, Springer (2012)
80. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL performance benchmark. In: ICDE. pp. 222–233. IEEE (2009)
81. Schneider, S., Lambers, L., Orejas, F.: Symbolic model generation for graph properties. In: International Conference on Fundamental Approaches to Software Engineering. pp. 226–243. Springer (2017)
82. Schölzel, H., Ehrig, H., Maximova, M., Gabriel, K., Hermann, F.: Satisfaction, restriction and amalgamation of constraints in the framework of m-adhesive categories. In: Proceedings Seventh ACCAT Workshop on Applied and Computational Category Theory, ACCAT 2012, Tallinn, Estonia, 1 April 2012. EPTCS, vol. 93, pp. 83–104 (2012)
83. Schonbock, J., Kappel, G., Wimmer, M., Kusel, A., Retschitzegger, W., Schwinger, W.: TETRABox - a generic white-box testing framework for model transformations. In: APSEC. pp. 75–82. IEEE (dec 2013)
84. Semeráth, O., Barta, Á., Horváth, Á., Szatmári, Z., Varró, D.: Formal validation of domain-specific languages with derived features and well-formedness constraints. Softw. Syst. Model. (2015)
85. Semeráth, O., Varró, D.: Graph constraint evaluation over partial models by constraint rewriting. In: ICMT. LNCS, Springer (2017)
86. Semeráth, O., Vörös, A., Varró, D.: Iterative and incremental model generation by logic solvers. In: FASE. pp. 87–103 (2016)
87. Sen, S., Baudry, B., Mottu, J.M.: On combining multi-formalism knowledge to select models for model transformation testing. In: ICST. pp. 328–337. IEEE (2008)
88. Spasic, M., Jovanovik, M., Prat-Pérez, A.: An RDF dataset generator for the Social Network Benchmark with real-world coherence. In: BLINK (2016)
89. Special C. of RTCA: DO-178C, software considerations in airborne systems and equipment certification (2012)
90. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
91. Szárnyas, G., Kővári, Z., Salánki, Á., Varró, D.: Towards the characterization of realistic models: Evaluation of multidisciplinary graph metrics. In: MODELS (2016)
92. Szárnyas, G., Izsó, B., Ráth, I., Varró, D.: The Train Benchmark: cross-technology performance evaluation of continuous model queries. Softw. Syst. Model. (2017)
93. Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., Goodwine, B., Baras, J.: Toward a science of cyber–physical system integration. Proc. of the IEEE 100(1), 29–44 (2012)
94. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In: TACAS, pp. 632–647. Springer (2007)
95. Torrini, P., Heckel, R., Ráth, I.: Stochastic simulation of graph transformation systems. In: FASE. LNCS, vol. 6013, pp. 154–157. Springer (2010)
96. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: Emf-incquery: An integrated development environment for live model queries. Sci. Comput. Program. 98, 80–99 (2015)
97. Varró, D., Bergmann, G., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z.: Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. Software and Systems Modeling 15(3) (2016)

98. Varró, D., Balogh, A.: The Model Transformation Language of the VIATRA2 Framework. Sci. Comput. Program. 68(3), 214–234 (October 2007)
99. Waltemath, D., et al.: Toward community standards and software for whole-cell modeling. IEEE Trans. on Bio-medical Engineering 63(10), 2007–14 (oct 2016)
100. Winkelmann, J., Taentzer, G., Ehrig, K., Küster, J.M.: Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars. Electr. Notes Theor. Comput. Sci. 211, 159–170 (2008)
101. Yakindu Statechart Tools: Yakindu, `http://statecharts.org/`
102. Zhang, J.W., Tay, Y.C.: GSCALER: synthetically scaling a given graph. In: EDBT. pp. 53–64 (2016), `https://doi.org/10.5441/002/edbt.2016.08`

A Further Definitions and Proofs

First, we show that the information ordering relation ($\sqsubseteq$) ensures the under- and over-approximation rules for any 3-valued truth value.

Lemma 1 (Information order vs Under- and over-approximation).
If X and Y are 3-valued logic values with $X \sqsubseteq Y$, then $(X = 1) \Rightarrow (Y = 1)$ and $(Y = 1) \Rightarrow (X \geq 1/2)$.

Proof. First, if $X = 1$ then according to the definition of information ordering, $(1 = 1/2) \vee (Y = 1)$ thus $Y = 1$.
Now if $Y = 1$ then similarly $(X = 1/2) \vee (X = 1)$ thus $X \geq 1/2$. □

Selected mathematical operations respect the information ordering:

Lemma 2 (Information order vs Mathematical operations).
If $X_1 \sqsubseteq Y_1, \ldots, X_n \sqsubseteq Y_n$ then

1. $1 - X_1 \sqsubseteq 1 - Y_1$
2. $min\{X_1, \ldots, X_n\} \sqsubseteq min\{Y_1, \ldots, Y_n\}$
3. $max\{X_1, \ldots, X_n\} \sqsubseteq max\{Y_1, \ldots, Y_n\}$

Proof. 1. Since $X_1 \sqsubseteq Y_1$ then either $X_1 = Y_1$ or $X_1 = 1/2$. If $X_1 = Y_1$, then $1 - X_1 = 1 - Y_1$ and therefore $1 - X_1 \sqsubseteq 1 - Y_1$ is true. Otherwise, if $X_1 = 1/2$, then $1 - X_1 = 1/2$ and $1/2 \sqsubseteq Y_1$ holds for any Y_1.

2. If some $X_i = 0$ then $Y_i = 0$. Thus $min\{X_1, \ldots, X_n\} = 0$ and $min\{Y_1, \ldots, Y_n\} = 0$, and $0 \sqsubseteq 0$ holds. Otherwise, if all $X_i = 1$ then all $Y_i = 1$, therefore $min\{X_1, \ldots, X_n\} = min\{Y_1, \ldots, Y_n\} = 1$, and $1 \sqsubseteq 1$ is satisfied. Finally, if there is no X_i with $X_i = 0$ but some $X_j = 1/2$ then $min\{X_1, \ldots, X_n\} = 1/2$, and $1/2 \sqsubseteq min\{Y_1, \ldots, Y_n\}$ holds for any $Y_1, \ldots, Y_n$ values.

3. If there is an $X_i = 1$, then $Y_i = 1$. Thus $max\{X_1, \ldots, X_n\} = 1$ and $max\{Y_1, \ldots, Y_n\} = 1$, and $1 \sqsubseteq 1$ holds. Otherwise, if all $X_i = 0$ then all $Y_i = 0$, therefore $max\{X_1, \ldots, X_n\} = max\{Y_1, \ldots, Y_n\} = 0$, and $0 \sqsubseteq 0$ is satisfied. Finally, if there is no X_i with $X_i = 1$, but some $X_j = 1/2$ then $max\{X_1, \ldots, X_n\} = 1/2$, and $1/2 \sqsubseteq max\{Y_1, \ldots, Y_n\}$ holds for any $Y_1, \ldots, Y_n$ values. □

Our the refinement relation respects information ordering for each formula φ.

Theorem 1. *Let P, Q be partial models with $P \sqsubseteq Q$ and φ be a graph pattern.*

- *If $[\![\varphi]\!]^P = 1$ then $[\![\varphi]\!]^Q = 1$; if $[\![\varphi]\!]^P = 0$ then $[\![\varphi]\!]^Q = 0$ (called **under-approximation**).*
- *If $[\![\varphi]\!]^Q = 0$ then $[\![\varphi]\!]^P \leq 1/2$; if $[\![\varphi]\!]^Q = 1$ then $[\![\varphi]\!]^P \geq 1/2$ (called **over-approximation**).*

Proof (Correctness of under- and over-approximation). Let φ be a graph pattern formula, and let P and Q be two partial models where $P \sqsubseteq Q$ with a refinement function $refine : Obj_P \to 2^{Obj_Q}$.

First, based on the definition of refinement, for each $p_1, p_2 \in Obj_P$ and $q_1 \in refine(p_1)$, $q_2 \in refine(p_2)$, the following statements hold for atomic predicates:

- $[\![\mathtt{C}(v)]\!]^P_{v \mapsto p_1} \sqsubseteq [\![\mathtt{C}(v)]\!]^Q_{v \mapsto q_1}$
- $[\![\mathtt{R}(v_1, v_2)]\!]^P_{v_1 \mapsto p_1, v_2 \mapsto p_2} \sqsubseteq [\![\mathtt{R}(v_1, v_2)]\!]^Q_{v_1 \mapsto q_1, v_2 \mapsto q_2}$
- $[\![v_1 \sim v_2]\!]^P_{v_1 \mapsto p_1, v_2 \mapsto p_2} \sqsubseteq [\![v_1 \sim v_2]\!]^Q_{v_1 \mapsto q_1, v_2 \mapsto q_2}$

Next, let φ_1 and φ_2 be two formulae, and let Z_1^P, Z_2^P, Z_1^Q and Z_2^Q be variable bindings with:

- $[\![\varphi_1]\!]^P_{Z_1^P} \sqsubseteq [\![\varphi_1]\!]^Q_{Z_1^Q}$ and $[\![\varphi_2]\!]^P_{Z_2^P} \sqsubseteq [\![\varphi_2]\!]^Q_{Z_2^Q}$,
- Z_1^P and Z_1^Q maps each variables of φ_1 to Obj_P and Obj_Q
- Z_2^P and Z_2^Q maps each variables of φ_2 to Obj_P and Obj_Q
- for all variables v in φ_1: $Z_1^Q(v) \in refine(Z_1^P(v))$
- for all variables v in φ_2: $Z_2^Q(v) \in refine(Z_2^P(v))$

Then the following refinements of formulae hold due to Lemma 2:

- $[\![\neg \varphi_1]\!]^P_{Z_1^P} = 1 - [\![\varphi_1]\!]^P_{Z_1^P} \sqsubseteq 1 - [\![\varphi_1]\!]^Q_{Z_1^Q} = [\![\neg \varphi_1]\!]^Q_{Z_1^Q}$
- $[\![\varphi_1 \wedge \varphi_2]\!]^P_{Z_1^P \cup Z_1^P} = min\{[\![\varphi_1]\!]^P_{Z_1^P}, [\![\varphi_2]\!]^P_{Z_2^P}\} \sqsubseteq min\{[\![\varphi_1]\!]^Q_{Z_1^Q}, [\![\varphi_2]\!]^Q_{Z_2^Q}\} = [\![\varphi_1 \wedge \varphi_2]\!]^Q_{Z_1^Q \cup Z_2^Q}$
- $[\![\varphi_1 \vee \varphi_2]\!]^P_{Z_1^P \cup Z_1^P} = max\{[\![\varphi_1]\!]^P_{Z_1^P}, [\![\varphi_2]\!]^P_{Z_2^P}\} \sqsubseteq max\{[\![\varphi_1]\!]^Q_{Z_1^Q}, [\![\varphi_2]\!]^Q_{Z_2^Q}\} = [\![\varphi_1 \vee \varphi_2]\!]^Q_{Z_1^Q \cup Z_2^Q}$
- $[\![\exists v : \varphi_1]\!]^P_{Z_1^P} = max\{[\![\varphi]\!]^P_{Z_1^P \cup v \mapsto p} : p \in Obj_P\} \sqsubseteq max\{[\![\varphi]\!]^Q_{Z_1^Q \cup v \mapsto q} : q \in refine(p)\} = [\![\exists v : \varphi_1]\!]^Q_{Z_1^Q}$
- $[\![\forall v : \varphi_1]\!]^P_{Z_1^P} = min\{[\![\varphi]\!]^P_{Z_1^P \cup v \mapsto p} : p \in Obj_P\} \sqsubseteq min\{[\![\varphi]\!]^Q_{Z_1^Q \cup v \mapsto q} : q \in refine(p)\} = [\![\forall v : \varphi_1]\!]^Q_{Z_1^Q}$

Since all these refinement relations hold, the statement of the theorem is now a direct consequence of Lemma 1. $\qquad\square$

Theorem 2 (Refinement operations ensure refinement). *Let P be a partial model and op be a refinement operation. If Q is the partial model obtained by executing op on P (formally, $P \xrightarrow{op} Q$) then $P \sqsubseteq Q$.*

Proof. We split the proof cases along the refinement operations. We investigate changes in the truth evaluation of different predicates implied by executing these operations, since each partial model is a refinement of itself if no changes occur.

- In case of *concretize*(p, val):
    - For each class predicate $p = \mathtt{C}_i(o)$, only operation *concretize*(p, val) can potentially change its value to 1 (or 0) if $[\![\mathtt{C}_i(o)]\!]^P = 1/2$. But then $[\![\mathtt{C}(o)]\!]^P = 1/2 \sqsubseteq [\![\mathtt{C}(o)]\!]^Q = 1$ (or $[\![\mathtt{C}(o)]\!]^Q = 0$), which satisfies the refinement relation.
    - Reasoning is identical for each reference predicate $\mathtt{R}(o_1, o_2)$.
    - An equivalence predicate $o_1 \sim o_2$ can be manipulated by operation *concretize*(p, val) to set an $1/2$ value to 1 (for self-loop equivalence predicates) or to either 1 or 0 (for non-self loops). In this case, the refinement conditions are trivally satisfied.

– When $splitAndConnect(o, mode)$ is applied then two o_1 and o_2 nodes of Q will be derived from a single node o in P.

- *At-least-two mode*:
  Since $[\![o \sim o]\!]^P = 1/2$ and both $[\![o_1 \sim o_1]\!]^Q = 1/2$ and $[\![o_2 \sim o_2]\!]^Q = 1/2$, but $[\![o_1 \sim o_2]\!]^Q = 0$, the refinement condition is satisfied.
- *At-most-two mode*:
  Since $[\![o \sim o]\!]^P = 1/2$ and both $[\![o_1 \sim o_1]\!]^Q = 1$ and $[\![o_2 \sim o_2]\!]^Q = 1$ while $[\![o_1 \sim o_2]\!]^Q = 1/2$, the refinement condition is satisfied.

$\square$

Corollary 1. *Let $P_0 \xrightarrow{op_1;...;op_k} P_k$ be an open derivation sequence of refinement operations wrt. φ. Then for each $0 \leq i \leq k$, $[\![\varphi]\!]^{P_i} \leq 1/2$.*

Proof. This is a direct consequence of Theorem 1. If we indirectly assume that $[\![\varphi]\!]^{P_k} \leq 1/2$ but $[\![\varphi]\!]^{P_i} = 1$ for some P_i along the derivation sequence, then all subsequent partial models P_j derived from P_i $(j > i)$ should be $[\![\varphi]\!]^{P_i} = 1$ which contradicts our assumption for $j = k$.

Corollary 2 (Soundness of model generation). *Let $P_0 \xrightarrow{op_1;...;op_k} P_k$ be a finite and open derivation sequence of refinement operations wrt. φ. If P_k is a concrete instance model M (i.e. $P_k = M$) then M is consistent (i.e. $[\![\varphi]\!]^M = 0$).*

Proof. We require that $[\![\varphi]\!]^{P_i} \leq 1/2$ for each i which includes the last partial model P_k. Since P_k is a concrete instance model, thus the 2-valued and 3-valued evaluation of φ must be identical (due to Proposition 1). Therefore $[\![\varphi]\!]^M = 1$ or $[\![\varphi]\!]^M = 0$, but only the latter case satisfies our assumption that $[\![\varphi]\!]^{P_k} \leq 1/2$. $\square$

Theorem 3 (Finiteness of model generation). *For any finite instance model M, there exists a finite derivation sequence $P_0 \xrightarrow{op_1;...;op_k} P_k$ of refinement operations starting from the most generic partial model P_0 leading to $P_k = M$.*

Proof (Sketch).
An instance model can always be generated:

1. Assume that M contains exactly n objects. Since P_0 consists of a single object, we need to create $n - 1$ new objects as part of the construction.
2. Execute action $splitAndConnect(o, mode)$ in *at-least-two* mode for $n - 1$ times, thus n (uncertain) objects will be available.
3. Concretize all $[\![o \sim o]\!]^{P_{n-1}} = 1$ and $[\![o_1 \sim o_2]\!]^{P_{n-1}} = 0$ (where $o_1 \neq o_2$).
4. Concretize all class and reference predicates in accordance with M by setting appropriate values in $concretize(p, val)$ to 1 or 0. As a result, P_{n-1} is gradually refined into a P_k which no longer contains an $1/2$ value, thus it is an instance model.

Model generation is always finite:

1. First, note that only $splitAndConnect(o, mode)$ actions are able to create new objects, $concretize(p, val)$ operations only fix values. Moreover, there are only finite number of uncertain values of p which still needs to be concretized.

2. The only recursive (thus potentially infinite) computation is carried out when action $splitAndConnect(o, mode)$ is executed in *at-least-two* mode.
3. Assume that in our computation, $splitAndConnect(o, mode)$ has been applied in *at-least-two* mode n times, thus P_n contains at least $n + 1$ objects, while our instance model has only n objects. We claim that this is a dead end derivation, thus we can cut it off and backtrack.
4. Due to the specification of the *at-least-two* model, all these objects are non-equivalent to each other, i.e. $[\![o_1 \sim o_2]\!]^{P_n} = 0$ for $o_1 \neq o_2$, thus they can never be merged during concretization. Now any consistent concretization of P_n will contain at least $n + 1$ different objects, which contradicts our indirect assumption that M has exactly n objects.

$\square$

Theorem 4 (Completeness of model generation). *For any finite and consistent instance model M with $[\![\varphi]\!]^M = 0$, there exists a finite open derivation sequence $P_0 \xrightarrow{op_1;...;op_k} P_k$ of refinement operations wrt. φ starting from the most generic partial model P_0 and leading to $P_k = M$.*

Proof. First, M is derivable by a finite derivation sequence due to Corollary 3. Now, for an indirect proof, let us assume that $[\![\varphi]\!]^M = 0$ yet there exist some partial model P_i along the finite derivation sequence $P_0 \xrightarrow{op_1;...;op_i} P_i \xrightarrow{op_{i+1};...;op_k} P_k$ where $[\![\varphi]\!]^{P_i} = 1$. However, the properties of under-approximation (in Theorem 1) imply that for all refinements P_j of P_i, $[\![\varphi]\!]^{P_j} = 1$. But since M is also a refinement of P_j (as each refinement operation ensures refinement, see Theorem 2), $[\![\varphi]\!]^M = 1$, which is a contradiction to our indirect assumption, thus it concludes the proof. $\square$

Theorem 5 (Decidability of model generation in finite scope). *Given a graph predicate φ and a scope $n \in \mathbb{N}$, it is decidable to check if a concrete instance model M exists with $|Obj_M| \leq n$ where $[\![\varphi]\!]^M = 0$.*

Proof (Sketch). While Theorem 4 ensures that there exists one finite derivation path, this does not directly guarantee that model generation would terminate along all derivation paths. Fortunately, the designated target scope n for the instance model implies an upper bound (i.e. scope) for the length of operation sequences that derive instance models of size n.

For any model M with n nodes and r edges, one can derive an operation sequences with n *splitAndConnect* operations followed by $r \cdot n^2$ *concretize* operations. Our refinement operations ensure that any derivation longer than $n + r \cdot n^2$ can be terminated as even smallest concrete instance model will exceed the target model scope n.

Corollary 3 (Incrementality of model generation). *Let us assume that no consistent models M^n exist for scope n, but there exists a larger consistent model M^m of size m (where $m > n$) with $[\![\varphi]\!]^{M^m} = 0$. Then M^m is derivable by a finite derivation sequence $P_i^n \xrightarrow{op_{i+1};...;op_k} P_k^m$ where $P_k^m = M^m$ starting from a partial model P_i^n of size n.*

Proof. As an indirect proof, let us assume that there exists a consistent model M^m of size m while there are no consistent models M^n up to scope n, but no derivation sequence $P_i^n \xrightarrow{op_{i+1};\ldots;op_k} P_k^m$ exists which would yield $M^m = P_k^m$ starting from a partial model P_i^n of size n.

Since M^m is consistent and finite, it is derivable thanks to the completeness theorem (Theorem 4) along some other derivation sequence $P_0 \xrightarrow{op_1;\ldots;op_l} P_k^m$ where $P_k^m = M^m$. Since each refinement operation used in $op_1;\ldots;op_l$ increases the size of P_i with at least one, the derivation sequence should reach a partial model P_j^n of size n.

With the trivial concretization (of turning all $1/2$ values to 1 for all class and reference predicates and to 0 for equivalence predicates), P_j^n can be turned into an instance model M_j^n which is also exactly of size n. Now if M_j^n is consistent, then our assumption is violated that no consistent models exist for scope n Otherwise, the tail of $P_j^n \xrightarrow{op_j;\ldots;op_l} P_k^m$ is a designated derivation sequence, which is a contradiction to our indirect assumption. $\qquad\square$

Corollary 4 (Completeness of refutation). *If all derivation sequences are closed for a given scope n, but no consistent model M^n exists for scope n for which $[\![\varphi]\!]^{M^n} = 0$, then no consistent models exist at all.*

Proof. As an indirect proof, let us assume that a consistent model M^m exists for some scope $m > n$, while all derivation sequences are closed for a given scope n and no consistent models M^n exist for that scope.

Since M^m is consistent and finite, then there shall be a derivation sequence $P_0 \xrightarrow{op_1;\ldots;op_m} P_m$ where $P^m = M^m$. However, all derivation sequences are closed for a given scope n, which holds for the prefix of this derivation sequence as well. Thus there shall be an intermediate partial model P_k along that sequence where (1) either no further refinement operations are executable or (2) φ has a match in P_k i.e. $[\![\varphi]\!]^{P_k} = 1$. In the former case, P^m would not be reachable by refinement operations. In the latter case, all refinements of P_k (including $P^m = M^m$) would have a match of φ due to Theorem 1. This is a contradiction which concludes our proof. $\qquad\square$

A.1 Multidimensional graph metrics

We use two graph metrics to show how realistic a model is, also used in our previous work on model analysis [91].

MPC The *multiplex participation coefficient* (*MPC*) [10] measures whether the references of an object $a \in Obj$ are uniformly distributed among reference types $\mathtt{R}_1, \ldots, \mathtt{R}_m$:

$$MPC(a) = \frac{|Obj|}{|Obj| - 1} \left[1 - \sum_{a \in Obj} \left(\frac{Degree(a, \{\mathtt{R}_i\})}{Degree(a, \{\mathtt{R}_1, \ldots, \mathtt{R}_m\})} \right)^2 \right],$$

where $Degree(a, \{R_1, \ldots, R_m\})$ denotes the total number of outgoing/incoming references of type $R_1, \ldots, R_m$ from/to object a.

$MPC(a)$ takes values in $[0, 1]$, equalling to 0 if all references of a belong to a single reference type, and to 1 if a has exactly the same number of references on each of reference types R_i.

Q *Pairwise multiplexity* (Q) [68] is defined for a pair of references typed $R_i, R_j \in R_1, \ldots, R_m$, where $1 \leq i, j \leq m$. Its value determines the ratio of objects from the model, which have reference instances in both references types R_i and R_j. Intuitively, the more mutual objects the two reference types have, the higher their pairwise multiplexity is.

The *node activity* binary vector NA_a $(a \in Obj)$ is defined as:

$$NA_a = \left\{ NA_a^{[R_1]}, NA_a^{[R_2]}, \ldots, NA_a^{[R_m]} \right\}, \text{where } NA_a^{[R_i]} = \exists o : R_i(a, o) \vee R_i(o, a),$$

Using this vector, the *pairwise multiplexity* metric is:

$$Q(R_i, R_j) = \frac{1}{|Obj|} \sum_{a \in Obj} NA_a^{[R_i]} NA_a^{[R_j]}.$$

$Q(R_i, R_j)$ takes values from the $[0, 1]$ interval, and equals to 1 if the activity vectors $NA_a^{[R_i]}$ and $NA_a^{[R_j]}$ are identical, i.e. when R_i and R_j belong to the same nodes.