Automated Generation of Consistent Graph Models with Multiplicity Reasoning

Kristóf Marussy, Oszkár Semeráth and Dániel Varró

Abstract—Advanced tools used in model-based systems engineering (MBSE) frequently represent their models as graphs. In order to test those tools, the automated generation of well-formed (or intentionally malformed) graph models is necessitated which is often carried out by solver-based model generation techniques. In many model generation scenarios, one needs more refined control over the generated unit tests to focus on the more relevant models. Type scopes allow to precisely define the required number of newly generated elements, thus one can avoid the generation of unrealistic and highly symmetric models having only a single type of elements. In this paper, we propose a 3-valued scoped partial modeling formalism, which innovatively extends partial graph models with predicate abstraction and counter abstraction. As a result, well-formedness constraints and multiplicity requirements can be evaluated in an approximated way on incomplete (unfinished) models by using advanced graph query engines with numerical solvers (e.g. IP or LP solvers). Based on the refinement of 3-valued scoped partial models, we propose an efficient model generation algorithm that generates models that are both well-formed and satisfy the scope requirements. We show that the proposed approach scales significantly better than existing SAT-solver techniques or the original graph solver without multiplicity reasoning. We illustrate our approach in a complex design-space exploration case study of collaborating satellites introduced by researchers at NASA Jet Propulsion Lab.

Index Terms—D.2.11.b Domain-specific architectures, E.1.d Graphs and networks, F.4.1.d Logic and constraint programming, I.6.4 Model Validation and Analysis

1 Introduction

Model-based systems engineering frequently uses complex modeling tools, like Capella, Artop, Matlab Simulink or Yakindu Statecharts. When these modeling tools are used in safety-critical systems, safety standards (like DO-330 [1] for avionics systems) may prescribe that (1) only the output of a *qualified tool* can be trusted, and (2) such a tool should meet the same requirements as the critical system component it designs. However, such quality assurance for the software running in modeling tools is very complex, which makes tool qualification an extremely costly process. As such, automated techniques for synthesizing effective test suites used in the software quality assurance of complex modeling tools would be highly beneficial.

The automated synthesis of high-quality test cases is a recurrent challenge in many areas of software and systems engineering in order to simultaneously improve quality and productivity. Since test cases created manually by engineers can easily miss important corner-cases of specifications, certain application areas (e.g. safety-critical software) substantially rely on such automated test case generators.

This paper focuses on *automated model generators* which represent tests in the form of graph models. This is a subclass of generators with high practical relevance but also high complexity. For example, graphs may models complex test stubs in object-oriented programs [2], [3] (e.g. nodes are objects, edges are pointers). The quality assurance of smart cyber-physical systems (CPS) can rely upon prototypical test contexts given in the form of graphs [4], [5], [6]. Model generators are also beneficial for testing modeling tools [7].

Further practical application scenarios are investigated in [8], which identifies a long-term research agenda aiming to provide desirable high-level properties for automated model generators. Using the terminology of [8], an advanced synthetic model generator should be domain-customizable, consistent, diverse, realistic and scalable.

For domain customizability, our paper uses precise underlying specification techniques to capture the domain concepts and their relations captured in the form of a metamodel, while *consistent* models can be further restricted by design rules or *well-formedness constraints* (defined as OCL constraints [9] or graph patterns [10], [11]).

There is a wide range of model generators such as Alloy [12], [13], Formula [14], [15], USE [16], UML2CSP [17], SDG [18], [19] and Viatra Solver [20], [21] to automatically derive *consistent* models for a given domain specification. Several generators are based on precise foundations offered by backend logic solvers (like SAT solvers [22], [23] or SMT solvers [24]). These tools excel at finding inconsistencies (if they exist) by interpreting domain specifications as a logic problem, but they can only derive small consistent models. Moreover, they fail to derive a *diverse* set of models [20], [25], which restricts their use in practical testing scenarios.

Alternatively, logic reasoning or search-based techniques can be lifted directly on the level of graphs [18], [19], [26] for model generation purposes. These approaches scale better with respect to the size and diversity of the derived models,

- *The authors are with the Department of Measurement and Infrormation Systems, Budapest University of Technology and Economics, Hungary and the MTA-BME Lendület Cyber-Phyisical Systems Research Group. E-mail: {marussy,semerath}@mit.bme.hu, daniel.varro@mcgill.ca*
- *Dániel Varró is with McGill University.*

but they may fail to reveal inconsistencies in specifications.

Finally, the *realistic* nature of synthetic models can also be important in test generation scenarios. For example, realistic test models used for autonomous cars represent real test environments [5], [6] while unrealistic test cases (e.g. obscure traffic situations) are considered as false positives. Failures caused by realistic scenarios are more severe, as they have more chance to happen on real workload. Several examples in *testing software-intensive CPSs* [5], [6], [18], [19], [27], [28] highlight this realistic aspect. Furthermore, the usability of automatically generated tests may be hindered by test cases that are not realistic (i.e., strange and difficult to comprehend for developers) [29].

Problem statement. To increase the realistic nature of models, one needs more refined control over the structure of the auto-generated models. For example, *partial snapshots* [30], [31] define model fragments that need to be extended by the model generator, thus it defines the expected *initial structure* of each models. Furthermore, *type scopes* [12] allow to precisely define the required number of newly generated elements (per type/class), thus focusing the generation process on more relevant instance models of the target domain.

While logic solver-based model generators support various scope constraints, they have severe scalability issues and they fail to generate complex graphs (without isolated nodes or star structures) with more than 50-70 nodes for complex domains [32]. The search-based approach [18], [19] can generate a large number of simple graph models with fine-grained type distributions, but it is unable to derive large *and* connected consistent models. Finally, the graph solver [32] can derive large and connected models, but it only allows to cap the total size of the model, and thus it is unable to fine-tune the models along type scopes.

Contributions. In order to improve the scalability and usefulness of automated model generation, we propose a novel technique that combines the advantages of partial model refinement techniques [26] with numeric reasoning on model scopes. In particular,

- We introduce *scoped partial models* as a background theory to represent type scopes for model generations.
- We define *a mapping of structural and well-formedness constraints into numeric constraints* that can be evaluated on scoped partial models.
- We use existing numerical solvers (i.e. IP and LP solvers) to efficiently guide the generator process.
- We extend an open source model generator [32] with type scope support and integrate various IP and LP solvers to provide a software prototype tool.
- We evaluate the effectiveness of the approach on numerous case studies including a running example of a complex design space exploration challenge [33] introduced by researchers at NASA Jet Propulsion Lab.

The current paper builds upon but substantially extends past research results in [8], [20], [32]. More specifically, the introduction and handling scope constraints are novel conceptual results of the current paper. In order to maintain the favorable theoretical properties (e.g. completeness, diversity) of the generic model generation framework formally proved in [8], [20], the refinement calculus is extended here to incorporate scopes. The prototype implementation builds on and extends [21], [32] by integrating various numerical solvers into the decision procedure. Finally, the experimental evaluation shows how novel results improve scalability and the realistic nature of models wrt. existing work.

Added value. With multiplicity reasoning, graph generators can be configured by numeric constraints to focus model generation on the relevant fragment of models. Although a single metric cannot ensure the realistic nature of models, but *ensuring the realistic distribution of model elements* were found to be useful in [18] as it filters out a wide range of surely unrealistic models. As such, automatically synthesized corner-cases will have higher practical relevance (e.g. test scenarios in autonomous driving will investigate relevant traffic situations).

With the help of numerical reasoning, graph generators will be able to measure and efficiently control the quantity of nodes. This significantly improves the performance of existing graph solver algorithms. Moreover, it enables a practical iterative workflow for test generation where initially, one can start with general scopes which are gradually refined to grow larger consistent models.

Finally, by adhering to the refinement calculus, the generator continues to provide favorable properties such as consistency, completeness or diversity (but the in-depth investigation of such properties is out of scope for the paper).

2 Models and partial models

The computational design synthesis of *interferometry mission architectures* has been introduced in [33] as a complex challenge for early mission planning for space missions of NASA where a designated architecture consists of collaborating satellites (of different size and capabilities) and radio communication between them. Each mission architecture involves multiple spacecrafts, which imposes an especially challenging design task. The authors of [33] suggested a technique to automatically enumerate promising design candidates with respect to the requirements, technical and resource constraints, and mission objectives. Since the original paper already used graph models and tools, we decided to adapt this as the running example of the paper.

In this section, we first provide foundations of domain-specific modeling languages (DSLs) and graph-based instance models formalized as partial models using relational logic enhanced with integer linear constraints.

2.1 Domain-specific modeling languages

A large set of industrial modeling tools (including e.g., Capella, Artop, Yakindu, Papyrus, etc.) use DSLs as conceptual foundation. The specification of a DSL typically starts from defining a *metamodel* (MM) and a set of *well-formedness constraints* (WF). A metamodel defines the main concepts and relations in a domain imposing the basic graph structure of *instance models*. WF constraints further restrict consistent (or valid) instance models of the language by defining additional design rules. In this paper, we use the Eclipse Modeling Framework (EMF) [34] metamodels and VIATRA well-formedness constraints [10], [11] as a technical foundation for domain modeling, which is also used in those industrial tools above as well as in [33]. Conceptually, the graph generation approach could be applied on other

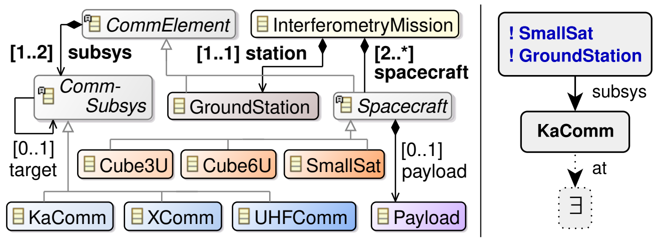


Fig. 1. Example metamodel and WF constraint error pattern

modeling formalisms too, e.g. UML Class Diagrams for defining the types and Object Constraint Language (OCL, [9]) for defining constraints as in [19], [35].

Example. The metamodel for interferometry constellation missions is shown in Fig. 1 using an EMF notation. An InterferometryMission consists of communicating Comm-Elements (as *EClasses*), which are equipped with Comm-Subsys subsystems (i.e., antennas with different communication frequencies) through their subsys *EReferences* for Ka, X, and UHF bands.

Spacecraft of different sizes, including cube satellites Cube3U and Cube6U, as well as small satellites SmallSat, may carry interferometry Payloads (photo sensors), and must be able to reach the GroundStation via radio links (to send sensor data) denoted by the target references.

As a foundation for generating consistent models first, we need a precise formal framework to specify DSLs for which purpose we rely on [7], [12], [14], [26], [30], [36].

Metamodels

The metamodel defines the main concepts and relations of the target domain.

Definition 1 (Metamodel). A metamodel defines a vocabulary $\Sigma = \{\mathsf{C}_1, \dots, \mathsf{C}_n, \varepsilon, \mathsf{R}_1, \dots, \mathsf{R}_m, \sim\}$ with unary predicate symbols C_i $(1 \leq i \leq n)$ defined for each *EClass*, a symbol ε denoting the existence of an object, a binary predicate symbol R_j $(1 \leq j \leq m)$ for each *EReference*, and a binary equivalence symbol $\sim$.

This formalism, in accordance with the EMF standard, handles references as relations: edges do not have identities and parallel edges of the same *EReference* are not allowed. Since our current work focuses on model generation for the structural part of graph models (i.e. nodes/objects and edges/links), we omit the detailed handling of attributes, which could be introduced similarly. Additionally, we introduce generator-specific concepts: a unary predicate ε denoting the existence of an object (in a normal model, each object is existing), and a binary predicate $\sim$ denoting the equivalence of objects (in a normal model, each objects are different from each other).

A metamodel also imposes several structural constraints on instance models to enforce syntactic consistency for model manipulation or model persistence operations:

(1) *Type Hierarchy (TH)* expresses that a more specific (child) class has every structural feature of the more general (parent) class;
(2) *Type Compliance (TC)* requires that for any relation $\mathsf{R}(o, t)$, its source and target objects o and t must have compliant types;
(3) *Abstract (ABS)*: If a class is defined as abstract, it is not allowed to have direct instances;
(4) *Multiplicity (MUL)* of structural features can be limited with upper and lower bound in the form of "lower..upper";
(5) *Inverse (INV)* states that two parallel references of opposite direction always occur in pairs.
(6) *Containment (CON)*: Instance models in EMF are expected to be arranged into a containment hierarchy, which is a directed tree along relations marked in the metamodel as containment (e.g., subsys or payload). The containment hierarchy is particularly relevant for serialization purposes.

Well-formedness constraints

In many industrial modeling tools, domain-specific WF constraints are defined by *error predicates* captured either as OCL invariants [9] or as graph patterns [10], [37]. A major practical subclass of such constraints can be formalized using first-order logic with transitive closure [26], [32], which can be efficiently evaluated by underlying query engines like [11] to validate models, or formally analyzed by model generators [7] to synthesize well-formed models.

Definition 2 (Syntax of graph predicate). A graph predicate φ is defined over a Σ vocabulary of a metamodel and an infinite set of (object) variables $\mathcal{V} = \{v_1, v_2, \dots\}$ using the following grammar rules:

$$\begin{array}{ll}
\varphi := \mathsf{C}(v) \mid \mathsf{R}(v_1, v_2) & \textit{type and reference pred.} \\
\quad \mid v_1 = v_2 & \textit{equivalence} \\
\quad \mid \neg\varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 & \textit{logic connectives} \\
\quad \mid \exists v : \varphi \mid \forall v : \varphi & \textit{quantified expression} \\
\quad \mid \varphi^{+}(v_1, v_2) & \textit{transitive closure}
\end{array}$$

Assuming that error patterns $\varphi_1, \dots, \varphi_n$ are defined for a domain, a model is consistent (or valid), if it does not satisfy any error predicates $\varphi_i(v_1, \dots, v_m)$, i.e. $\neg\exists v_1, \dots, v_m : \varphi_i(v_1, \dots, v_m) = \forall v_1, \dots, v_m : \neg\varphi_i(v_1, \dots, v_m)$.

Error predicates $\varphi_1, \dots, \varphi_8$ in the *satellite* case study capture the following design rules of interferometry missions.

- A CommElement may only have a single transmitting subsys (the other subsys, if present, may only receive):
$$\varphi_1(e) := \exists c_1, c_2 \colon \mathsf{subsys}(e, c_1) \land \mathsf{subsys}(e, c_2) \land c_1 \neq c_2 \land (\exists t \colon \mathsf{target}(c_1, t)) \land (\exists t \colon \mathsf{target}(c_2, t)).$$
- The GroundStation can only receive and may not have any outgoing communication links:
$$\varphi_2(g) := \exists c, t \colon \mathsf{GroundStation}(g) \land \mathsf{subsys}(g, c) \land \mathsf{target}(c, t).$$
- At least two different Spacecrafts must have the interferometry Payload configured:
$$\varphi_3 := \forall s_1, s_2 \colon \neg(\exists p \colon \mathsf{payload}(s_1, p)) \lor \neg(\exists p \colon \mathsf{payload}(s_2, p)) \lor s_1 = s_2.$$
- All Spacecraft must have a communication path (transitive closure of radio links) to the GroundStation:

$$\begin{aligned} \mathit{link}(s_1, s_2) &\coloneqq \exists c_1, c_2 \colon \mathsf{subsys}(s_1, c_1) \land \mathsf{subsys}(s_2, c_2) \\ &\qquad \land \mathsf{target}(c, s_2), \\ \varphi_4(s) &\coloneqq \mathsf{Spacecraft}(s) \\ &\qquad \land (\forall g \colon \neg\mathsf{GroundStation}(g) \lor \neg \mathit{link}^+(s, g)). \end{aligned}$$

- There may be no communication loops, i.e., communication paths from a CommElement to itself:

$$\varphi_5(e) \coloneqq \mathit{link}^+(e, e).$$

- CommSubsystems can only communicate if they use the same frequency band:

$$\begin{aligned} \varphi_6(c_1, c_2) \coloneqq{} & \mathsf{target}(c_1, c_2) \\ & \land \neg(\mathsf{KaComm}(c_1) \land \mathsf{KaComm}(c_2)) \\ & \land \neg(\mathsf{XComm}(c_1) \land \mathsf{XComm}(c_2)) \\ & \land \neg(\mathsf{UHFComm}(c_1) \land \mathsf{UHFComm}(c_2)). \end{aligned}$$

- Cube3U satellites can only cross-link (send data to another satellite) using an UHFComm transmitter, but can only communicate with the GroundStation using a XComm transmitter:

$$\begin{aligned} \varphi_7(s) \coloneqq \exists c_1, c_2, e \colon & \mathsf{Cube3U}(s) \land \mathsf{subsys}(s, c_1) \\ & \land \mathsf{subsys}(e, c_2) \land \mathsf{target}(c_1, c_2) \\ & \land \neg(\mathsf{UHFComm}(c_1) \land \mathsf{Spacecraft}(e)) \\ & \land \neg(\mathsf{XComm}(c_1) \land \mathsf{GroundStation}(e)). \end{aligned}$$

- Only a SmallSat or the GroundStation may be configured with a KaComm subsystem:

$$\begin{aligned} \varphi_8(e) \coloneqq{} & (\exists s \colon \mathsf{subsys}(e, s) \land \mathsf{KaComm}(s)) \\ & \land \neg\mathsf{SmallSat}(e) \land \neg\mathsf{GroundStation}(e). \end{aligned}$$

The error predicate φ_8 is depicted on the right side of Fig. 1 as a graph pattern using the graphical syntax of the GROOVE graph transformation tool [38].

Because the structural constraints on metamodels can be formalized as WF constraints [7] using the graph predicate language of [26], [32], we can evaluate both kinds of constraints uniformly with first-order logic. However, as structural constraints are prevalent in modeling tasks, in the following, we will exploit their special structure, especially that of *MUL* and *CON* constraints, to speed up model generation by numerical reasoning, while retaining full support for arbitrary WF constraints.

Type scopes

To guide model generation towards more relevant models in a domain, type scopes are frequently used to specify the number of required elements of each type (class). For example, Alloy [12] introduces *scope bounded* analysis for relational specifications. For larger models, prescribing lower and upper bounds may ensure realistic distribution of types in auto-generated test cases and benchmarks.

Type scope constraints define lower and upper bounds for the number of instances generated for a specific class. A *lower type scope constraint* $L_i \le \mathsf{C}_i$ and an *upper type scope constraint* $\mathsf{C}_i \le U_i$ respectively assert that there are at least L_i and at most U_i instances of the class C_i (where $L_i, U_i \in \mathbb{N}$). We require that a generated model must satisfy the conjunction of all scope constraints of a given type.

Test and benchmark generation tasks require models of some finite size n, whereas for proving the inconsistency of modeling languages, cases up to a small size n are checked according to the small scope assumption [12]. Therefore, we assume the existence of an upper bound n on the number of objects in the generated models, which can be seen as a type scope bound on a common supertype of all types.

Our formulation of type scopes extends the notation of scopes introduced in Alloy [12], which supports only upper ($\mathsf{C}_i \le U_i$) and exact limits ($\mathsf{C}_i = E_i$) (but not lower bounds). Alloy also limits type scopes and type hierarchy. If a type scope is specified for a class C_i, its supertypes cannot have a type scope. Scopes in Alloy cannot express problems where the sums of (upper) type scope bounds do not coincide with the number of objects ($\sum_i U_i > n$), because the model size n can only be specified as a type scope bound on the common supertype of all types. Therefore, these upper scope constraints and all lower scope constraints need to be formulated as additional constraints instead.

Given the constraints $30 \le \mathsf{Spacecraft}$, $\mathsf{Spacecraft} \le 50$, and $\mathsf{SmallSat} \le 15$ for our running example, generated models may contain between 30 and 50 Spacecrafts. Moreover, at most 15 of these spacecrafts can be SmallSats.

2.2 Scoped partial models

In this paper, we introduce the concept of *3-valued scoped partial models* as an extension of partial models proposed in [32]. The goal of partial models is to explicitly represent uncertainty in models, thus a single partial model represents a set of potential (traditional) instance models. We combine two techniques to capture uncertainty in a partial model. First, *3-valued logic* is used to explicitly represent uncertain *structural properties* of models with a third $1/2$ (unspecified or unknown) truth value (besides 1 and 0, which stand for *true* and *false*) in accordance with [8], [26], [39]. Secondly, *quantitative information* is attached to the partial model to precisely represent the *known (or required) size* of the models. Later, we use partial models as states of model generation to represent intermediate solutions with uncertain parts denoted with truth-value $1/2$ and its size.

From a formal perspective, the first partial modeling technique implements *predicate abstraction* [26], [40] on graph models, while the second technique provides *counter abstraction* [41], [42] on the nodes of the graph model.

Definition 3 (3-valued partial model). A *3-valued scoped partial model* is a tuple $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$, where $\mathcal{O}_P$ is a finite set of individuals in the model (i.e., the objects), $\mathcal{I}_P(S) \colon \mathcal{O}_P^{\alpha(S)} \to \{0, 1, 1/2\}$ provides a 3-values interpretation for all *structural predicate symbols* $S \in \Sigma$ (where $\alpha(S)$ is the arity of the predicate symbol S), and the *object scopes* $\mathcal{S}_P$ define numerical constraints over $\mathcal{O}_P$.

Structural predicates

First, let us discuss the interpretation of structural predicate symbols ($\mathcal{I}_P$) of partial models.

- **Node (class) predicates:** $\mathcal{I}_P$ gives a 3-valued interpretation of each class symbol C_i in Σ: $\mathcal{I}_P(\mathsf{C}_i) \colon \mathcal{O}_P \to \{1, 0, 1/2\}$ that gives if it is true, false, or unspecified if an object is an instance of a class C_i.
- **Edge (reference) predicates:** $\mathcal{I}_P$ gives a 3-valued interpretation to each reference symbol R_j in Σ: $\mathcal{I}_P(\mathsf{R}_j) \colon \mathcal{O}_P \times \mathcal{O}_P \to \{1, 0, 1/2\}$, that gives if it is true, false, or unspecified if there is a reference R_j between two objects.

TABLE 1
Explanation for existence and self-equivalence predicates

$\varepsilon(x)$	$x \sim x$	Description	Symbol	Regularity criteria
1	1	concrete object	[1..1]	$\mathcal{S}_P \vDash \widehat{x} = 1$
1/2	1	uncertain, concrete	[0..1]	$\mathcal{S}_P \vDash \widehat{x} \leq 1$
1	1/2	multi-object	[1..*]	$\mathcal{S}_P \vDash \widehat{x} \geq 1$
1/2	1/2	uncertain, multi	[0..*]	unrestricted

- **Existence predicate:** $\mathcal{I}_P$ gives 3-valued interpretation $\mathcal{I}_P(\varepsilon)\colon \mathcal{O}_P \to \{1, 1/2\}$ to the ε and predicates. $\mathcal{I}_P(\varepsilon)(x) = 1$ and $1/2$ means certain or possible existence of object x.
- **Equivalence predicate:** $\mathcal{I}_P$ also gives 3-valued interpretation $\mathcal{I}_P(\sim)\colon \mathcal{O}_P \times \mathcal{O}_P \to \{1, 0, 1/2\}$ to the $\sim$ predicate. $\mathcal{I}_P(\sim)(x,y) = 1$, 0, and $1/2$ mean that it is true, false, or unknown whether x and y are equal.

In the context of model generation, we restrict the possible combination of those predicates to exclude inconsistent and irrelevant constructs that are not productive as intermediate states of model generation.

Definition 4 (Structural Regularity). A partial model is structurally regular if it satisfies the following criteria:

- *Object merges* are impossible, i.e., distinct objects $x \not\equiv y$ ($x, y \in \mathcal{O}_P$) are surely not equal: $\mathcal{I}_P(\sim)(x,y) = 0$.
- There are no *unmerged equivalent objects*: if $x, y \in \mathcal{O}_P$ and $\mathcal{I}_P(\sim)(x,y) = 1$, then $x \equiv y$.
- There are no *nonexistent objects*, i.e., an object $x \in \mathcal{O}_P$ cannot be surely nonexistent: $\mathcal{I}_P(\varepsilon)(x) \neq 0$.

Table 1 summarizes the possible cases of uncertain existence and self-equivalence.

Fig. 2 shows three partial models. Truth values of *class predicates* are denoted by labels on nodes (missing labels correspond to 0 values). *Reference predicates* with 1 and 1/2 values are denotes as solid and dashed arrows, respectively. Nodes with Dashed borders correspond to 1/2 values of the *existence* ε predicate. Uncertain *equivalences* are shown with dashed $\sim$ loops, but to reduce clutter, certain self-equivalences are not depicted. Thus, multi-objects have dashed borders and dashed $\sim$ loops and concrete objects are shown with solid borders.

In P_0 (on the left side of Fig. 2), the multi-object new_{3U} (with uncertain existence and self-equivalence) is certainly of type Cube3U, but not of type CommSubsys.

Object scopes

Next, let us discuss the numerical constraints in a partial model ($\mathcal{S}_P$ in $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$). $\mathcal{S}_P$ defines a system of linear inequalities over variables $\mathcal{V}_P = \{\widehat{x} \mid x \in \mathcal{O}_P\}$ associated with the nodes $\mathcal{O}_P$of the partial model P.

Definition 5 (Assignment, Solution, Entailment). An assignment k maps each variable $\widehat{x} \in \mathcal{V}_P$ to a non-negative integer $k\colon \mathcal{V}_P \to \mathbb{N}$. If an assignment k satisfies the system of linear inequalities of $\mathcal{S}_P$, then it is called a *solution* of $\mathcal{S}_P$ (written as $k \vDash \mathcal{S}_P$). We write $S_1 \vDash S_2$ for the *entailment* of linear constraints, i.e., when every solution $k \vDash S_1$ also satisfies $k \vDash S_2$.

The values of these variables represent the number of concrete nodes that a single abstract node represents. If $k\colon \mathcal{V}_P \to \mathbb{N}$ is a solution, then partial model P may represent a concrete instance model M where each $x \in \mathcal{O}_P$ stands for exactly $k(\widehat{x})$ objects.

Partial models P_0 and P_1 illustrated in Fig. 2 define two systems of linear inequalities ($\mathcal{S}_{P_0}$ and $\mathcal{S}_{P_1}$ respectively) over the same three variables: $\widehat{new_{\mathsf{3U}}}$, $\widehat{new_{\mathsf{X}}}$ and $\widehat{new_{\mathsf{UHF}}}$. In $\mathcal{S}_{P_0}$, the linear equation $\widehat{new_{\mathsf{3U}}} + \widehat{new_{\mathsf{X}}} + \widehat{new_{\mathsf{UHF}}} = 10$ ensures that P_0 represent instance models with exactly 10 objects. A potential variable assignment $k\colon \widehat{new_{\mathsf{3U}}} \mapsto 4, \widehat{new_{\mathsf{X}}} \mapsto 3, \widehat{new_{\mathsf{UHF}}} \mapsto 3$ is a possible solution of both $\mathcal{S}_{P_0}$ and $\mathcal{S}_{P_1}$, and represent models with 4 3U, 3 X and 3 UHF objects. As $\mathcal{S}_{P_1}$ contains more constraints than $\mathcal{S}_{P_0}$, $\mathcal{S}_{P_1} \vDash \mathcal{S}_{P_0}$ is holds trivially. However, $\mathcal{S}_{P_0} \vDash \mathcal{S}_{P_1}$ is not true as $k'\colon \widehat{new_{\mathsf{3U}}} \mapsto 10, \widehat{new_{\mathsf{X}}} \mapsto 0, \widehat{new_{\mathsf{UHF}}} \mapsto 0$ is a solution for $\mathcal{S}_{P_0}$ but not for $\mathcal{S}_{P_1}$.

The *regularity criteria* of scoped partial models ensures consistency of $\mathcal{I}_P$ and the object scopes $\mathcal{S}_P$.

Definition 6 (Numerical regularity). A partial model P is numerically regular, if $\mathcal{S}_P$ is satisfiable, and for each object $x \in \mathcal{O}_P$:

$$\begin{aligned} \mathcal{I}_P(\sim)(x,x) = 1 &\quad\Rightarrow\quad \mathcal{S}_P \vDash [\widehat{x} \leq 1] \\ \mathcal{I}_P(\varepsilon)(x) = 1 &\quad\Rightarrow\quad \mathcal{S}_P \vDash [\widehat{x} \geq 1] \end{aligned}$$

Therefore $\mathcal{S}_P$ carries *at least as precise* numerical information about the (multi-)objects as $\mathcal{I}_P$. Table 1 summarizes the possible combinations of existence, equivalence and scopes.

A partial model is *regular* if it is both structurally and numerically regular. In the following, all partial models will be assumed to be regular.

2.3 Refinement and concretization of PMs

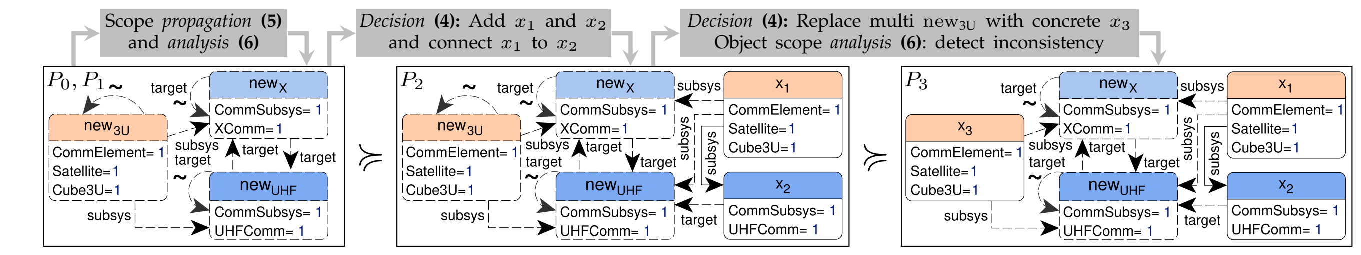
We carry out model generation along a sequence of refinement steps that derive new partial models by increasing their size but gradually reducing the level of uncertainty in each model while continuously checking (an approximated version of) well-formedness and scope constraints. Thus we introduce the formal concept of refinement for scoped partial models which simultaneously refines both the 3-valued logic structure and the system of linear inequalities.

First, during refinement, unknown $1/2$ values are refined to either 0 or 1, according to the *refinement ordering relation*.

Definition 7 (Logic value refinement). A truth value Y is a refinement of X (formally $X \succcurlyeq Y$), where either $X = 1/2$ as it is refined into $Y = 1$ or 0, or $X = Y$ remains unchanged: $X \succcurlyeq Y := (X = 1/2) \vee (X = Y)$.

Logic refinement is defined between the logic structures associated with partial model, where some $1/2$ values in the interpretation $\mathcal{I}_P$ of a partial model P is refined into either 1 or 0 values. Informally, during refinement between structurally regular partial models, (i) objects with $1/2$ values for $\sim$ may be split into multiple objects, (ii) objects with $1/2$ values for ε may disappear, (iii) and class or reference predicates with $1/2$ values are refined to 1 or 0.

Definition 8 (Logic structure refinement). Given an *abstraction* function $abs : \mathcal{O}_Q \to \mathcal{O}_P$, a logic interpretation $\mathcal{I}_Q$ of a partial model Q refines a logic interpretation $\mathcal{I}_P$ of P (denoted by $\mathcal{I}_P \succcurlyeq_{abs} \mathcal{I}_Q$) if for each n-ary


Scope *propagation* (5) and *analysis* (6)
Decision (4): Add x_1 and x_2 and connect x_1 to x_2
Decision (4): Replace multi new_{3U} with concrete x_3 Object scope *analysis* (6): detect inconsistency
P_0, P_1
P_2
P_3
new3U
newX
newUHF
x1
x2
x3
CommElement= 1
Satellite= 1
Cube3U= 1
CommSubsys= 1
XComm= 1
UHFComm= 1
target
subsys
~

$$\mathcal{S}_{P_0}\colon\ \widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} = 10 \quad (\#n)$$

$$\begin{aligned}\mathcal{S}_{P_1}\colon\ \widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} &= 10\\ 5 \le \widehat{new_X}+\widehat{new_{UHF}} & \quad (\mathrm{sp}_1)\\ \widehat{new_{3U}} \le \widehat{new_X}+\widehat{new_{UHF}} & \quad (\mathrm{sp}_2)\\ \widehat{new_X}+\widehat{new_{UHF}} \le 2\widehat{new_{3U}} & \quad (\mathrm{sp}_3)\\ \left.\begin{array}{c}4 \le \widehat{new_{3U}} \le 4, 0 \le \widehat{new_X} \le 6,\\ 0 \le \widehat{new_{UHF}} \le 6\end{array}\right\} & \quad (\mathrm{sa})\end{aligned}$$

$$\begin{aligned}\mathcal{S}_{P_2}\colon\ \widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} &= 8\\ 4 &\le \widehat{new_X}+\widehat{new_{UHF}}\\ \widehat{new_{3U}} &\le \widehat{new_X}+\widehat{new_{UHF}}\\ \widehat{new_X}+\widehat{new_{UHF}} &\le 2\widehat{new_{3U}}+1\\ 3 \le \widehat{new_{3U}} \le 3,\ & 0 \le \widehat{new_X} \le 5\\ 0 \le \widehat{new_{UHF}} \le 5,\ & \widehat{x_1} = 1, \widehat{x_2} = 1\end{aligned}$$

$$\begin{aligned}\mathcal{S}_{P_3}\colon\ \widehat{new_X}+\widehat{new_{UHF}} &= 7\\ 4 &\le \widehat{new_X}+\widehat{new_{UHF}}\\ \widehat{new_X}+\widehat{new_{UHF}} &\le 3\\ 0 \le \widehat{new_X} \le 5,\ & 0 \le \widehat{new_{UHF}} \le 5\\ \widehat{x_1} = 1, \widehat{x_2} &= 1, \widehat{x_3} = 1\end{aligned}$$

$7 \le 3$ is UNSAT $\rightarrow$ backtrack

Fig. 2. Scoped partial models and their refinements. Linear equation systems were simplified by carrying out substitutions for conciseness.

predicate symbol S $\in \Sigma$ (type, reference, equivalence and existence) and for each $o_1, \ldots, o_n \in \mathcal{O}_Q$:

$$\mathcal{I}_P(\mathsf{S})(abs(o_1), \ldots, abs(o_n)) \succcurlyeq_{abs} \mathcal{I}_Q(\mathsf{S})(o_1, \ldots, o_n).$$

Moreover, existing objects cannot be removed:

$$\forall p \in \mathcal{O}_P : \varepsilon(p) \Rightarrow (\exists q \in \mathcal{O}_Q : abs(q) = p).$$

During refinement, the linear inequality systems are also refined with respect to the entailment relation. Informally, during the refinement of $\mathcal{S}_P$ into $\mathcal{S}_Q$, it (i) may split some of variables into the sum of multiple variables (e.g., all occurrences of a variable $\widehat{x}$ in $\mathcal{S}_P$ are replaced with $\widehat{x_1} + \widehat{x_2} + \widehat{x_3}$ in $\mathcal{S}_Q$), and (ii) it may induce stricter constraint over the variables (e.g., $\widehat{x} \le 3$ is refined to $1 \le \widehat{x} \le 2$).

Definition 9 (Linear inequality system refinement). Given an abstraction function $abs : \mathcal{O}_Q \to \mathcal{O}_P$, a linear linear inequality system $\mathcal{S}_Q$ of partial model Q is a refinement of $\mathcal{S}_P$ of P (denoted by $\mathcal{S}_P \succcurlyeq_{abs} \mathcal{S}_Q$) if $\mathcal{S}_Q \vDash \mathcal{S}_P^{abs}$, where $\mathcal{S}_P^{abs}$ denotes the system of linear inequalities obtained from $\mathcal{S}_P$ by replacing every occurrence of each variable $\widehat{x} \in \mathcal{O}_P$ with $\sum\{\widehat{y} \mid abs(y) = x\}$.

In this paper, we define the refinement of 3-valued scoped partial models using 3-valued scoped partial models using simultaneous logic structure refinement and linear inequality refinement. Conceptually, each model generation step will carry out such a refinement thus making the model larger but less uncertain.

Definition 10 (Partial model refinement). A 3-valued partial model $Q = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mathcal{S}_Q \rangle$ refines a partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ (denoted as $P \succcurlyeq Q$) if there is an abstraction function $abs\colon \mathcal{O}_Q \to \mathcal{O}_P$ where

$$\mathcal{I}_P \succcurlyeq_{abs} \mathcal{I}_Q \text{ and } \mathcal{S}_P \succcurlyeq_{abs} \mathcal{S}_Q.$$

If a 3-valued partial model M only contains 1 and 0 values, then M represents a traditional (concrete) *instance model*. In an instance model, $\mathcal{S}_M \vDash \widehat{x} = 1$ for all $x \in \mathcal{O}_M$ due to numerical regularity, i.e., each object is concrete.

Fig. 2 depicts three refinements $P_0 \succcurlyeq P_1$, $P_1 \succcurlyeq P_2$, and $P_2 \succcurlyeq P_3$. P_0 and P_1 have the same object set ($\mathcal{O}_{P_0} = \mathcal{O}_{P_1}$) and graph structure. Therefore, the abstraction function $abs_1\colon \mathcal{O}_{P_1} \to \mathcal{O}_{P_0}$ is the identity function. Compared to $\mathcal{S}_{P_0}$, $\mathcal{S}_{P_1}$ contains an additional linear equation. Every solution of $\mathcal{S}_{P_0}$ is also a solution of $\mathcal{S}_{P_1}$, which ensures $P_0 \succcurlyeq P_1$.

The abstraction function $abs_2\colon \mathcal{O}_{P_2} \to \mathcal{O}_{P_1}$ maps new_{3U}, new_X, new_{UHF} to the objects in P_1 with the same identifiers, while $abs_2(x_1) = new_{3U}$ and $abs_2(x_2) = new_{UHF}$. The objects x_1 and x_2 were *split* from new_{3U} and new_{UHF}, respectively. To obtain $\mathcal{S}_{P_2}$, we replaced each occurrence of $\widehat{new_{3U}}$ and $\widehat{new_{UFH}}$ with $\widehat{new_{3U}} + \widehat{x_1}$ and $\widehat{new_{UFH}} + \widehat{x_2}$. Furthermore, the constant 1 replaces occurrences of $\widehat{x_1}$ and $\widehat{x_2}$, because $\widehat{x_1} = \widehat{x_2} = 1$ (x_1 and x_2 are concrete objects). As there are no new linear equations, $\mathcal{S}_{P_2}$ is otherwise equivalent to $\mathcal{S}_{P_2}$.

In $P_2 \succcurlyeq P_3$, x_3 replaces the multi-object new_{3U} with x_3, i.e., $abs_3(x_3) = new_{3U}$, while all other objects of P_3 are mapped to the object with the same name in P_2.

Refinement is transitive, i.e., if $P_1 \succcurlyeq P_2$ and $P_2 \succcurlyeq P_3$ with the abstraction functions $abs_1\colon \mathcal{O}_{P_2} \to \mathcal{O}_{P_1}$ and $abs_2\colon \mathcal{O}_{P_3} \to \mathcal{O}_{P_2}$, then $P_1 \succcurlyeq P_3$ with the abstraction function $abs_1 \circ abs_2$. Hence after a chain of refinements $P_0 \succcurlyeq P_1 \succcurlyeq \cdots \succcurlyeq M$, we may obtain a concrete model M. Such a refinement chain will be constructed during model generation.

2.4 Predicate evaluation over partial models

While constraints expressed in first-order graph logic with transitive closure can be easily evaluated over concrete graph models (with true or false outcome), the evaluation of graph predicates over partial models naturally has a 3-valued semantics.

Definition 11 (Semantics of graph predicates). The semantics of a graph predicate $\varphi(v_1, \ldots, v_n)$ over a partial model P with variable binding Z is denoted with $[\![\varphi(v_1, \ldots, v_n)]\!]^P_Z$, and defined as follows:

$$\begin{aligned}[\![\mathsf{C}(v)]\!]^P_Z &:= \mathcal{I}_P(\mathsf{C})(Z(v))\\ [\![\mathsf{R}(v_1, v_2)]\!]^P_Z &:= \mathcal{I}_P(\mathsf{R})(Z(v_1), Z(v_2))\\ [\![\varepsilon(v)]\!]^P_Z &:= \mathcal{I}_P(\varepsilon)(Z(v))\\ [\![v_1 = v_2]\!]^P_Z &:= \mathcal{I}_P(\sim)(Z(v_1), Z(v_2))\\ [\![\neg\varphi]\!]^P_Z &:= 1 - [\![\varphi]\!]^P_Z\end{aligned}$$

$$\begin{aligned}
[\![\varphi_1 \land \varphi_2]\!]^P_Z &\coloneqq min\,([\![\varphi_1]\!]^P_Z, [\![\varphi_2]\!]^P_Z)\\
[\![\varphi_1 \lor \varphi_2]\!]^P_Z &\coloneqq max([\![\varphi_1]\!]^P_Z, [\![\varphi_2]\!]^P_Z)\\
[\![\exists v : \varphi]\!]^P_Z &\coloneqq min\,\{[\![\varepsilon(v) \land \varphi]\!]^P_{Z,v\mapsto o} \mid o \in \mathcal{O}_P\}\\
[\![\forall v : \varphi]\!]^P_Z &\coloneqq max\{[\![\neg\varepsilon(v) \lor \varphi]\!]^P_{Z,v\mapsto o} \mid o \in \mathcal{O}_P\}\\
[\![\varphi^+(v_1, v_2)]\!]^P_Z &\coloneqq max\{[\![\exists m_1, \dots, m_n :\\
&\qquad R(v_1, m_1) \land \dots \land R(m_n, v_2)]\!]^P_Z \mid n \in \mathbb{N}\}
\end{aligned}$$

Note that graph predicates can be *approximately evaluated* directly on partial models by predicate rewriting [26] without materializing all potential concrete models.

When an error predicate φ evaluates to true along variable binding Z, i.e. $[\![\varphi]\!]^P_Z = 1$ then this binding Z is called a *match* of φ in P. If an error predicate φ has a match in P, then this violation is already a proof of inconsistency.

Definition 12 (Structural (in)consistency of a partial model). Given a partial model P and a set of error predicates $\varphi_1, \dots, \varphi_k$, a partial model P is *structurally inconsistent* if there exists φ_i and a binding Z where φ_i has a match in P, i.e. $[\![\varphi_i]\!]^P_Z = 1$. A partial model is *structurally consistent* if $[\![\varphi_i]\!]^P_Z = 0$ for all φ_i and Z.

Due to the incompleteness, a partial model can potentially be neither structurally consistent, nor structurally inconsistent during partial model refinement. However, our model generation approach can avoid inconsistent partial solutions during model generation by *approximation of predicates*, so the consistency can be checked before a concrete instance model is obtained.

2.4.1 Approximation of logic predicates

In [8], [32], we defined over- and under-approximations of predicates over partial models to drive the model generation process along meaningful refinements. If an error predicate φ is surely satisfied in a partial model P ($[\![\varphi]\!]^P_Z = 1$, *under-approximation* of errors), then no concrete instance model M obtained from P by a refinement $P \succcurlyeq M$ can be structurally consistent [8]. Thus, partial model P can be safely dropped from the set of candidate intermediate solutions without discarding any valid instance models, and model generation needs to continue along a different refinement chain.

Theorem 1 (Forward refinement of predicates [8]). Let $\varphi(v_1, \dots, v_k)$ be a logic expression, P and Q partial models, where $P \succcurlyeq Q$ through $abs\colon \mathcal{O}_Q \to \mathcal{O}_P$, and $Z\colon \{v_1, \dots, v_k\} \to \mathcal{O}_Q$ a variable binding.

- If $[\![\varphi]\!]^P_{abs\circ Z} = 1$, then $[\![\varphi]\!]^Q_Z = 1$.
- If $[\![\varphi]\!]^P_{abs\circ Z} = 0$, then $[\![\varphi]\!]^Q_Z = 0$.

One can establish a dual *over-approximation* property for the validity of Q, which ensures that no valid model will marked as invalid (and vice versa):

Theorem 2 (Backward refinement of predicates [8]). Let $\varphi(v_1, \dots, v_k)$ be a logic expression, P and Q partial models, where $P \succcurlyeq Q$ through $abs\colon \mathcal{O}_Q \to \mathcal{O}_P$, and $Z\colon \{v_1, \dots, v_k\} \to \mathcal{O}_Q$ a variable binding.

- If $[\![\varphi]\!]^Q_Z = 1$, then $[\![\varphi]\!]^P_{abs\circ Z} \geq 1/2$.
- If $[\![\varphi]\!]^Q_Z = 0$, then $[\![\varphi]\!]^P_{abs\circ Z} \leq 1/2$.

2.4.2 Approximation of scope constraints

In scoped partial models, analogous properties hold for the constraints imposed on multi-objects by object scopes $\mathcal{S}_P$. For that purpose, we introduce the notation $\#^{1/2}_v[\![\varphi]\!]^P_Z$ to capture the number of concrete objects and multi-objects that *may* satisfy φ. Moreover, $\#^{1}_v[\![\varphi]\!]^P_Z$ represents the number of those that *must* satisfy φ. In a concrete model, these two formulas coincide, and they are equal to the number of concrete objects that satisfy φ.

Definition 13 (Number of matching objects). Given a logic formula $\varphi(u_1, \dots, u_k, v)$ and variable binding $Z\colon \{u_1, \dots, u_k\} \to \mathcal{O}_P$ (which only excludes v),

$$\#^{1/2}_v[\![\varphi]\!]^P_Z \coloneqq \sum\{\widehat{x_i} \mid x_i \in \mathcal{O}_P, [\![\varphi]\!]^P_{Z,v\mapsto x_i} \geq 1/2\}$$

denotes the sum of scope variables $\widehat{x_i}$ associated with objects x_i that may satisfy φ. Analogously,

$$\#^{1}_v[\![\varphi]\!]^P_Z \coloneqq \sum\{\widehat{x} \mid x \in \mathcal{O}_P, [\![\varphi]\!]^P_{Z,v\mapsto x} = 1\}$$

is the sum of scope variables associated with objects that surely satisfy φ.

For example, if $\{x_1, \dots, x_m\} = \{x_i \in \mathcal{O}_P \mid [\![\varphi]\!]^P_{Z,v\mapsto x_i} \geq 1/2\}$ are the objects that possibly satisfy φ, then the linear inequality $L \leq \widehat{x_1} + \dots + \widehat{x_m} \leq U$ can be written as $L \leq \#^{1/2}_v[\![\varphi]\!]^P_Z \leq U$.

In P_2 in Fig. 2, $\#^{1/2}_s[\![\exists c : \mathsf{subsys}(s, c)]\!]^{P_2} = \widehat{new_{3U}} + \widehat{x_1}$ is the linear expression for the number of objects that may have an outgoing subsys reference. $\#^{1}_s[\![\exists c : \mathsf{subsys}(s, c)]\!]^{P_2} = \widehat{x_1}$ is the number of objects that surely have an outgoing subsys reference. (The empty variable binding $Z = \emptyset$ was omitted from the notation for conciseness.)

Now we can evaluate type scope bounds on partial models checking linear inequalities on the objects scopes in a partial model P.

Definition 14 (Scope (in)consistency of a partial model). Given a partial model P and a set of type scope bounds $\{L_i \leq \mathsf{C}_i \leq U_i \mid i = 1, \dots, m\}$, P is *scope inconsistent* if there exists a type scope bound $L_i \leq \mathsf{C}_i \leq U_i$ such that $\mathcal{S}_P \vDash \#^{1/2}_v[\![\mathsf{C}_i(v)]\!]^P < L_i$ or $\mathcal{S}_P \vDash \#^{1}_v[\![\mathsf{C}_i(v)]\!]^P > U_i$. P is *scope consistent* if $\mathcal{S}_P \vDash \#^{1}_v[\![\mathsf{C}_i(v)]\!]^P \geq L_i$ and $\mathcal{S}_P \vDash \#^{1/2}_v[\![\mathsf{C}_i(v)]\!]^P \leq U_i$ for all $i = 1, \dots, m$.

A partial model can potentially by neither scope consistent, nor scope inconsistent during partial model refinement. On a concrete model M, $\mathcal{S}_M \vDash \#^{1/2}_v[\![\mathsf{C}_i(v)]\!]^P$ and $\mathcal{S}_M \vDash \#^{1}_v[\![\mathsf{C}_i(v)]\!]^P$ coincide, and correspond the to the number of objects of type $\mathsf{C}i$. Hence scope consistent concrete models indeed satisfy all type scope bounds.

Now we can over- and under-approximate scope constraints on partial models and maintain scope consistency during model generation as follows:

Theorem 3 (Forward refinement of scopes). Let φ be a logic expression, and P and Q partial models where $P \succcurlyeq Q$ through the abstraction function $abs\colon \mathcal{O}_Q \to \mathcal{O}_P$, and $L, U \in \mathbb{Z}$. Then the following implications hold:

$$\begin{aligned}
\mathcal{S}_P \vDash \#^{1/2}_v[\![\varphi]\!]^P_{abs\circ Z} < L &\implies \mathcal{S}_Q \vDash \#^{1/2}_v[\![\varphi]\!]^Q_Z < L, && \text{(i)}\\
\mathcal{S}_P \vDash \#^{1}_v[\![\varphi]\!]^P_{abs\circ Z} > U &\implies \mathcal{S}_Q \vDash \#^{1}_v[\![\varphi]\!]^Q_Z > U, && \text{(ii)}
\end{aligned}$$

i.e., (i) when objects that *may* satisfy φ violate a lower bound L in P, they also violate it in any refined partial model Q, and (ii) objects that *must* satisfy φ similarly carry forward the violation of the upper bound U.

Therefore, if a partial model P is scope inconsistent, it can be safely dropped from the set of potential intermediate solutions, as all of its refinements remain scope inconsistent.

Dually, if Q is scope consistent $P \succcurlyeq Q$, then P cannot be scope inconsistent. This statement, formalized below, is the over-approximation of validity for scope constraints.

Theorem 4 (Backward refinement of scopes). Let φ be a logic expression, and P and Q partial models where $P \succcurlyeq Q$ along the abstraction function $abs\colon \mathcal{O}_Q \to \mathcal{O}_P$, and $L, U \in \mathbb{Z}$. Then the following implications hold:

$$\mathcal{S}_Q \models \#_v^{1}[\![\varphi]\!]_Z^Q \geq L \implies \mathcal{S}_P \not\models \#_v^{1/2}[\![\varphi]\!]_{abs\circ Z}^P < L,$$
$$\mathcal{S}_Q \models \#_v^{1/2}[\![\varphi]\!]_Z^Q \leq U \implies \mathcal{S}_P \not\models \#_v^{1}[\![\varphi]\!]_{abs\circ Z}^P > U.$$

The forward and backward refinement properties enable the generation of structurally and scope consistent models along partial model refinements, where WF and scope constraints are approximately checked. Theorems 1–4, as discussed in Section 3.6, ensure the correctness and completeness of the process.

3 Model generation with scope reasoning

In this section we exploit numerical information present in object scopes of PMs to efficiently generate large instance models that satisfy type scope bounds, as well as structural and WF constraints. We combine techniques from *advanced graph query processing*, *SAT solving* and *integer programming* to tackle the scalability problems of existing graph generation approaches.

As the core conceptual contribution of the current paper, we combine the evaluation of relational constraints and numerical reasoning with object scopes by propagating information between 3-valued logic interpretation and objects scopes of the partial model. The intuition behind this idea is that while constraints expressed as object scopes are not as expressive as those captured in relational logic, dedicated numerical solvers allow earlier detection of constraint violations by considering the global effects of all constraints on the number of objects in the generated instance models at the same time. Therefore, the evaluation of the original WF constraints on the partial models and the scope analysis are complementary to each other.

As a summary, object scopes will allow early detection of partial models that cannot be completed to an instance model due to the inappropriate number (e.g., too few or too many) of objects, while WF constraint evaluation will enforce more complex structural validation rules.

3.1 Model generation process

We propose a model generation process (shown in Fig. 3) based on partial models with object scopes that can exploit the numeric information present in scoped partial models. The generation starts from an initial partial model, which is gradually refined until it obtains a concrete model satisfying the generation objectives defined by the number of required objects and the WF constraints. Thus, the generator explores the state space formed by partial models that are reachable by refinement, which ensures that isomorphic states are explored only once. Our generator has the following components:

- The *initial partial model* **(1)** is the starting point of the generation, which express type scope constraints as object scopes. It is either set to the most general (maximally underspecified) partial model or to a partial snapshot model provided by an engineer which is to be extended by the generator. The other inputs of the generator are the *type scope bounds* **(2)** and the *structural and WF constraints* **(3)** to be satisfied by the generated models.
- *Refinement operators* include decision rules, unit propagation rules and scope propagator rules to obtain new PMs from already discovered ones.
  - *Decisions* add new information to the PM and *unit propagations* enforce the necessary consequences of decisions by evaluating structural and WF constraints using the 3-valued interpretation. For decisions and unit propagations, we reuse the set of operators **(4)** defined by the *GraphSolver* (GS) [32] (which were proved to be sound and complete).
  - *Scope propagators* **(5)** restrict object scopes according to type scope bounds, structural, and WF constraints. This gives an opportunity for numerical reasoning with the new object scope information.
- *Object scope analysis* **(6)** performs numerical reasoning using state-of-the-art integer programming (IP) and linear programming (LP) techniques on object scopes. The results of numerical reasoning are fed back to the partial model and the best-first search strategy.
- 3-valued logic semantics **(7)** are exploited to *over-* and *under-approximate* WF constraint violations. PMs that cannot be repaired by refinement (i.e., which surely violate a constraint) are discarded by backtracking.
  - The constraint evaluation component uses an *efficient, incremental graph query engine* [11], [43] to ensure the scalability of this step.[1]
  - Unsatisfiable objects scopes, which are caused by type scope or WF constraint violations that cannot be repaired by refinement, are discarded by *backtracking* according to Theorems 1–4. Detecting these violations as early as possible is crucial for reducing the traversed state space.
- Isomorphic PMs reached by different refinements are detected by *state coding* **(8)** based on graph shapes [47], which ensures that isomorphic states are explored only once.
- *Heuristic* best-first search **(9)** combined with *backjumping* and *random restarts* preferentially investigates PMs that can be quickly refined into valid concrete models. Object scope analysis allows selecting such PMs more accurately than existing approaches that only rely on

1. The incremental graph query engine requires in-place updates to the partial model, which is (technologically) limited to be single-threaded. Nevertheless, it is possible to parallelize incremental query evaluation [44], [45], as well as to maintain several partial models for parallel state space exploration [46]. Integrating these improvements to the solver is in the scope of future work.

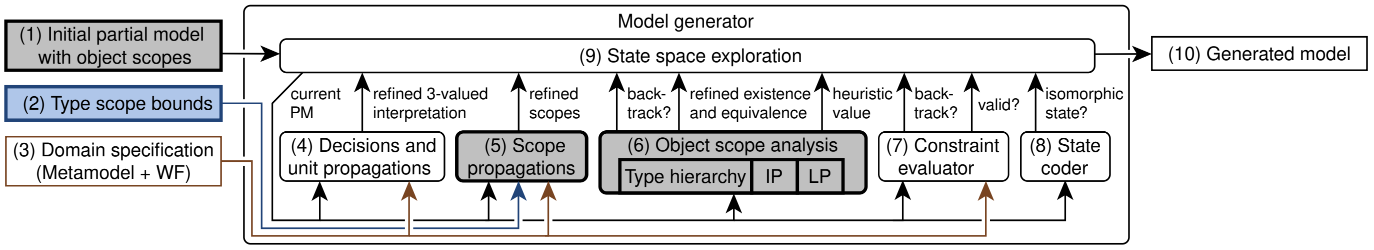

(1) Initial partial model with object scopes
(2) Type scope bounds
(3) Domain specification (Metamodel + WF)
Model generator
(9) State space exploration
current PM
refined 3-valued interpretation
refined scopes
back-track?
refined existence and equivalence
heuristic value
back-track?
valid?
isomorphic state?
(4) Decisions and unit propagations
(5) Scope propagations
(6) Object scope analysis
Type hierarchy
IP
LP
(7) Constraint evaluator
(8) State coder
(10) Generated model

Fig. 3. Block diagram of the model generator. Blocks interacting with object scopes are shaded for emphasis.

3-valued interpretation.

- When a structurally (Definition 12) and scope consistent (Definition 14) concrete model **(10)** is found, it is recorded as output. The generation is either terminated, or (if additional models are desired) the search is resumed after backtracking (as if the found solution was invalid). For the collected outputs, the solution management features of *GraphSolver*, which can ensure the diversity of the models [20], can be leveraged.

Next, we discuss the key novel components of our generator in more details.

3.2 Initial scoped PM

Model generation starts from an initial partial model P_{init}, which is a common abstraction of all possible concrete instance models of the metamodel. In P_{init},

- there is an object new_i for each non-abstract class C_i, i.e., $\mathcal{O}_{P_{init}} = \{new_i \mid \mathsf{C}_i \in \Sigma\}$;
- for each C_i, new_i is multi, i.e., $\mathcal{I}_{P_{init}}(\varepsilon)(new_i) = 1/2$ and $\mathcal{I}_{P_{init}}(\sim)(new_i, new_i) = 1/2$; and
- $\mathcal{I}_{P_{init}}(\mathsf{C}_i)(new_i) = 1$ (new_i is an instance of C_i).

Other class C_j and reference R_k predicates are set to 1 or 0 wherever required by type hierarchy and conformance constraints. Otherwise they are set to $1/2$.

The object scopes $\mathcal{S}_{P_{init}}$ in the initial PM introduce a variable $\widehat{new_i}$ for each class C_i, which allows expressing type scope bounds directly.

If model generation extends an initial partial snapshot, it can also be incorporated into P_{init}. For each given object x_i, $\mathcal{S}_{P_{init}}$ contains the equality $\widehat{x_i} = 1$ to mark x_i as a concrete object with exactly one instance. Interpretation of type and reference predicates between given objects are set in accordance with the initial partial snapshot, while reference predicates leading between new objects are $1/2$.

The partial model P_0 in Fig. 2 is a fragment of the initial partial model P_{init} for generating instances of the *satellite* metamodel in Fig. 1. The multi-objects new_{3U}, new_{X}, and new_{UHF} correspond to the classes $\mathsf{Cube3U}$, XComm, and $\mathsf{UHFComm}$. In $\mathcal{S}_{P_0} = \{\widehat{new_{3U}} + \widehat{new_{3U}} + \widehat{new_{3U}} = 10\}$, the linear equation (marked as #n in Fig. 2) encodes that models with exactly 10 objects shall be generated.

3.3 Scope propagation

Scope propagation refines the partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ into a new partial model $P \succcurlyeq Q = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \cup S \rangle$, where S is a set of linear inequalities deduced from P, type scope bounds, as well as structural and WF constraints. Because the inequalities are necessary consequences of the constraints, every consistent concrete model $P \succcurlyeq M$, satisfies them. Therefore each consistent instance model M is also a refinement of Q.

Table 2 summarizes the rules used to deduce linear inequalities implied by type scope bounds and structural metamodel constraints from the partial model. In the table, the relation $\leq$ refers to the usual *implication order* $0 \leq 1/2 \leq 1$ of truth values (and not the *refinement order* $\succcurlyeq$).

Fig. 2, the linear equations (sp$_1$-sp$_3$) in $\mathcal{S}_{P_1}$ were obtained from P_0 and the type scope bound $5 \leq \mathsf{CommSubsys}$ by scope propagation. The *lower type scope bound* $\mathsf{CommSubsys}$ implies (sp$_1$). By applying the *containment hierarchy, lower* and *upper bound* rules to the containment (CON) reference subsys [1..2] according to the multiplicity (MUL) bounds defined in Fig. 1, yielding the linear equations (sp$_2$) and (sp$_3$)

Other WF constraints which have numerical consequences can also be translated to object scopes by adding object scope constraints corresponding to lower and upper bounds of the number of objects allowed by the constraint.

Consider the error predicate $\varphi_8(e) \coloneqq (\exists s \colon \mathsf{subsys}(e, s) \wedge \mathsf{KaComm}(s)) \wedge \neg \mathsf{SmallSat}(e) \wedge \neg \mathsf{GroundStation}(e)$. Because subsys is a containment (CON) reference, φ_8 enforces that each KaComm instance be contained in a $\mathsf{SmallSat}$ or a $\mathsf{GroundStation}$. Due to the upper multiplicity (MUL) bound of 2, for each $\mathsf{SmallSat}$ or $\mathsf{GroundStation}$, there may be no more than 2 KaComm instances. We obtain the following scope propagation rule as the linearization of φ_8:

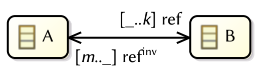
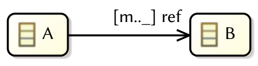
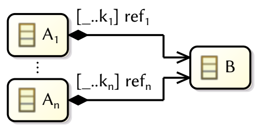
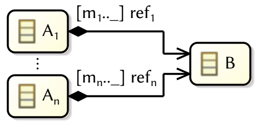
$$\#_u^1 [\![\mathsf{KaComm}(u)]\!]^P \leq \\ 2 \cdot \#_v^{1/2} [\![\mathsf{SmallSat}(v) \vee \mathsf{GroundStation}(v)]\!]^P.$$

In our current implementation, the user needs to manually provide linear inequality versions of well-formedness constraints to exploit them during object scope propagation. A higher level of automatization seems feasible (similarly as in [48]) and is in the scope of future work.

3.4 Object scope analysis

Object scope analysis is responsible for numerical reasoning with object scope constraints, which guides model generation and refines the interpretation $\mathcal{I}_P$. The refined relations may allow applying further unit and scope propagation operators, which in turn are opportunities for further scope analysis. The analysis requires efficient maintenance and solution of linear constraints.

TABLE 2
Scope propagation rules for type scope bounds and structural constraints

Constraint / class diagram	Description	Linear inequality
$L_i \leq \mathsf{C}_i$	**Lower type scope bound.** There must be at least L_i instances of C_i in the concrete model. Hence objects that *may* be C_i ($\mathcal{I}_P(\mathsf{C}_i)(x) \geq 1/2$) must represent at least L_i concrete objects.	$L_i \leq \#_v^{1/2}[\![\mathsf{C}_i(v)]\!]^P$
$\mathsf{C}_i \leq U_i$	**Upper type scope bound.** There may be at most U_i instances of C_i in the concrete model. Hence object that *must* be C_i ($\mathcal{I}_P(\mathsf{C}_i)(x) = 1$) may represent at most U_i concrete objects.	$\#_v^{1}[\![\mathsf{C}_i(v)]\!]^P \leq U_i$
	**Upper bound with inverse lower bound.** Each A may be connected to at most k B instances by the reference ref, and each B must be connected to at least m A instances by the inverse ref$^{\text{inv}}$. Hence there can be at most k B instances for each possible m A instances.	$m \cdot \#_u^{1}[\![\mathsf{B}(u)]\!]^P \leq k \cdot \#_v^{1/2}[\![\mathsf{A}(v)]\!]^P$
	**Lower bound.** Each A instance must be connected to at least m B instances by the reference ref. Hence for each existing A instance, potential targets of ref must represent at least m concrete objects.	$m \leq \#_v^{1/2}[\![\mathsf{ref}(u,v)]\!]^P_{u\mapsto x}$ for all $x \in \mathcal{O}_P$, $[\![\varepsilon(u) \wedge \mathsf{A}(u)]\!]^P_{u\mapsto x} = 1$
	**Containment hierarchy, upper bound.** Let $\mathsf{A}_1, \dots, \mathsf{A}_n$ be the possible containers of B. For every $i = 1, \dots, n$, instances of the class A_i can contain at most k_i instances of B (infinite upper bounds $k_i = *$ are replaced by a suitably large finite constant K). Hence for each possible instance of each A_i, there may be no more than k_i instances of B.	$\#_u^{1}[\![\mathsf{B}(u)]\!]^P \leq \sum_{i=1}^{n} k_i \cdot \#_v^{1/2}[\![\mathsf{A}_i(v)]\!]^P$
	**Containment hierarchy, lower bound.** Let $\mathsf{A}_1, \dots, \mathsf{A}_n$ be the possible containers of B. For every $i = 1, \dots, n$, each instance of the class A_i must contain at least m_i instances of B. Hence for each instance of each A_i, there must be at least m_i possible instances of B.	$\sum_{i=1}^{n} m_i \cdot \#_u^{1}[\![\mathsf{A}_i(u)]\!]^P \leq \#_v^{1/2}[\![\mathsf{B}(v)]\!]^P$

Linear constraint maintenance. As the size of the partial model P grows, the number of variables and constraints in $\mathcal{S}_P$ may also grow. Two techniques reduce the size of $\mathcal{S}_P$ to improve analysis. Firstly, concrete objects always stand for a single object ($\mathcal{S}_P \vDash x = 1$ if x is concrete). Instead of explicitly storing coefficients of a variable x for each concrete object and linear constraint, occurrences of x are replaced with the constant 1. Thus, the number of variables equals to the number of multi-objects, which usually does not grow during model generation.

Secondly, redundant linear inequalities are eliminated to prevent the number of scope constraints from growing indefinitely, exploiting the following two properties: (i) In our decision and scope propagation rules [32], no new multi-objects are added to the partial model. (ii) In our object scope analysis rules, the coefficients of multi-object variables only depend on the meta-model. This results in many pairs of constraints of the form $L_1 \leq \alpha_1\widehat{x_1} + \dots + \alpha_n\widehat{x_n} \leq U_1$ and $Ł_2 \leq \alpha_1\widehat{x_1} + \dots + \alpha_n\widehat{x_n} \leq U_2$, which can be replaced by $max\{L_1, L_2\} \leq \alpha_1\widehat{x_1} + \dots + \alpha_n\widehat{x_n} \leq min\{U_1, U_2\}$.

Numerical reasoning. Numerical reasoning carried out by object scope analysis (i) discovers refinements of the existence ε and equivalence $\sim$ relations implied by the object scopes, (ii) initiates backtracking on unsatisfiable object scopes, and (iii) calculates a heuristic for guiding the search based on the number of objects required to finish the model.

Scopes are analyzed to find lower and upper bounds of object scope variables x associated with each object $x \in \mathcal{O}_x$. If the lower bound is positive ($\mathcal{S}_P \vDash \widehat{x} \geq 1$), x represents at least one object and cannot be removed from P. We set $\mathcal{I}_P(\varepsilon)(x) = 1$ to record this fact. If the upper bound is 1 ($\mathcal{S}_P \vDash \widehat{x} \leq 1$), x represents at most one object, implying $\mathcal{I}_P(\sim)(x, x) = 1$. Lastly, an upper bound of 0 ($\mathcal{S}_P \vDash \widehat{x} \leq 0$) means x can be removed from P.

If a contradiction is detected when obtaining variable bounds, there is no instance model represented by the scoped PM P. The generator discards P and backtracks.

Otherwise, the sum of lower bounds is used as a heuristic in best-first search to approximate the number of decisions still required to obtain a valid instance model. This heuristic prefers the creation of smaller models when possible. However, due to the randomized state-space exploration, it does not guarantee models of minimum size.

In Fig. 2, the linear equations (sa), which represent feasible lower and upper bounds of the object scopes, were obtained by scope analysis of $\mathcal{S}_{P_1} \setminus \{(\text{sa})\}$.

Scope analysis of P_3 detects an inconsistency (highlighted in red in Fig. 2) caused by the unsatisfiable object scopes $\mathcal{S}_{P_3}$. No refinement of P_3 is a valid instance model. Therefore P_3 can be safely discarded by backtracking.

3.5 Scope analysis methods

We propose three methods for the reasoning, which are shown in Table 3. The *Type hierarchy based* analysis can only handle linear equations derived from type scope bounds. It is a quick preliminary step that can detect some contradictions early without invoking an external solver. Analysis with *Integer Programming* (IP) and *Linear Programming* (LP) *solvers* is considerably more precise, and handles any linear equations. However, the invocation of the external solver may be costly, especially in the case of IP, which is NP-complete. In Section 4, we compare the effectiveness of these approaches.

Type hierarchy based scope analysis analyses linear equations coming from type scope bounds, which are always of the form $L_i \leq \widehat{x_1} + \dots + \widehat{x_k} \leq U_i$ (Table 2). Exploiting that all variables in $\mathcal{S}_P$ are nonnegative, the inequalities $L_i \leq \widehat{x_1} + \dots + \widehat{x_k} \leq U_i$ and $L_j \leq \widehat{x_1} + \dots + \widehat{x_k} + \dots + \widehat{x_m} \leq U_j$, which are formed when C_i is a subtype of C_j, can be

TABLE 3
Scope analysis methods

	Type scope	Structural	Other WF
Type hierarchy	●	○	○
IP solver	●	●	●
LP solver	●	●	●

Legend: ○ = not supported, ● = supported

replaced with $L_j \leq \widehat{x_1} + \cdots + \widehat{x_k} \leq min\{U_i, U_j\}$ and $max\{L_i, L_j\} \leq \widehat{x_1} + \cdots + \widehat{x_k} + \cdots + \widehat{x_m} \leq U_j$. This process is performed for each pair of compatible inequalities until no more bounds can be tightened. Contradiction is detected when the lower bound of some inequality becomes larger than the upper bound.

Integer programming solvers are used for scope analysis by translating the object scope constraints into an IP problem and repeatedly solving for lower and upper bounds of variables. Formally, for all object $x \in \mathcal{O}_P$, the problems

$$x_{min} = min\ \widehat{x}, \quad \text{s.t. } \mathcal{S}_P, \quad \forall y \in \mathcal{O}_P : \widehat{y} \in \mathbb{N}, \qquad x_{max} = max\ \widehat{x}, \quad \text{s.t. } \mathcal{S}_P, \quad \forall y \in \mathcal{O}_P : \widehat{y} \in \mathbb{N}$$

are solved and the inequality $x_{min} \leq \widehat{x} \leq x_{max}$ is added to $\mathcal{S}_P$. Results of solver calls are cached to reduce invocations.

Linear programming. By replacing the set of natural numbers $\mathbb{N}$ with the nonnegative reals $\mathbb{R}^{\geq 0}$, the LP *relaxation* of the problem is obtained. In contrast with IP, LP can be solved in polynomial time. However, the obtained bounds for scope variables may not be as accurate, and opportunities for backtracking or refinement of the ε and $\sim$ predicates may be detected later. In order to detect these opportunities as early as possible, we rely on the fact that the number of object represented by a multi-object is always an integer. When the relaxation produces an inexact solution with non-integer x_{min} or x_{max}, the solution is rounded to assert the constraint $\lceil x_{min} \rceil \leq \widehat{x} \leq \lfloor x_{max} \rfloor$.

3.6 Correctness and completeness

As the main benefit of 3-valued PMs, a *multi-object* may represent multiple separate, unequal *concrete* objects in an instance model. As such, even sets of very large instance models can be abstracted by a small PM, which enables the model generator to use a concise representation of their state as a scoped partial model.

Based on Section 2.4, logic constraints can be approximately evaluated over intermediate solutions. Forward- and backward approximation theorems Theorems 1 and 3 ensure that if a partial model violates a WF or scope constraint, all refinements of that intermediate solution will also surely violate it, thus it can be safely discarded. WF and scope constraints are also directly evaluated on all finished (concrete) models, thus ensuring the *correctness* of the approach, i.e. all generated models are instances of the metamodel, and satisfy all WF and scope constraints.

Additionally, according to Theorems 2 and 4, if there is a valid concretization of an intermediate model, the partial model will not be discarded due to WF and scope constraint violations. In a bounded scope, all valid partial models will be considered [8]. Therefore, the approach is *complete* within a bounded scope (i.e., when models up to a finite size are sought) and it will explore all valid solutions.

While multiplicity reasoning can greatly increase the performance the model generator, the descriptive power of ordinary PMs is *limited to linear constraints*. This limits the multiplicity reasoning on simple scopes, but ensures that the numerical problems can be efficiently solved in each step of model generation.

4 Experimental evaluation

We carried out an experimental evaluation of generating consistent instance models with multiplicity reasoning provided by object scopes to address the following research questions:

RQ1 How effective are the different scope analysis techniques for model generation in terms of execution time?
RQ2 How does our approach scale in execution time on satisfiable problems. . .
  RQ2.1 . . . in the presence of type scope bounds?
  RQ2.2 . . . with unbounded type scopes?
RQ3 How does our approach scale in execution time on unsatisfiable problems?
RQ4 To what extent can type scope bounds help in generating models with realistic type distributions?

4.1 Domains

Due to the absence of systematically constructed performance benchmarks for the evaluation model generation for DSLs, we evaluated our approach in the context of 3 different domains (and the corresponding DSLs) that include complex structural and WF constraints. The first domain served as the running example in this paper (Fig. 1):

- **Sat** is the design space exploration challenge introduced by researchers at NASA Jet Propulsion Lab [33]. As a specific characteristic of this case study, structural constraints specify the number of CommSubsystems and Payloads that can be fitted to a number of Spacecraft, while WF constraints encode additional design rules concerning the satellite communication network.

Two additional case studies exemplify test generation scenarios for industrial modeling tools:

- **Sct:** *Yakindu* is an industrial modeling environment for statecharts [49]. This scenario represents generating tests for a concrete modeling tool (Yakindu Statecharts). The WF constraints of the language help avoiding common semantic errors (e.g., the lack of an Entry object signifying state). As a specific characteristic of this case study, most constraints can participate in object scope propagation (after linearization) to determine the possible numbers of objects (e.g., the Entry and its outgoing Transition instance that denotes the initial state).
- **Met:** *Ecore* is the meta-modeling language of EMF [34]. This scenario represents test generation for a modeling framework (e.g., code generation and persistency). As a specific characteristic, while this case study uses a large number of classes in a complex inheritance hierarchy along with WF constraints, only few of them can be translated into linear inequalities for scope propagation.

In addition to their practical relevance, the these two languages have been used as case studies by multiple model generation papers [32], [36], [50], [51], [52], [53].

4.2 Scope analysis methods

Setup. *This experiment aims at determining which scope analysis method should test engineers use for scalable model generation.* We generated models containing up to 100 objects in the **Sat** domain.

The original *GraphSolver* (**GS/O**) served as a baseline (with type scopes translated to WF constraints). We evaluated the *Type Hierarchy* (**GS/S/TH**) scope analysis method, which relies on no external solver, in addition to the *Integer Programming* (**GS/S/IP**) and *Linear Programming* (**GS/S/LP**) methods. We selected external solvers widely used in industry and research from the *COIN-OR* suite: *COIN-OR Branch and Cut* v2.9.9 for **GS/S/IP** and *COIN-OR Linear Programming Solver* v1.16.10 for **GS/S/LP**[2].

As there were no manually created models available for **Sat**, type scope bounds derived from engineering expectation. When specifying the type scope bounds, we ensured that they were satisfiable, i.e., a valid model exists with the specified number of objects. Unsatisfiable scope bounds are quickly detected by the IP/LP sovers, but cause other approaches to explore a very large number of partial models.

A timeout of 5 minutes was set for each model generation with increasing model sizes. Runs for a given model size were repeated 30 times to account for variance caused by the random exploration and backjumping employed in the generator, as well as the runtime environment.

We also accounted for warm-up effects and memory handling of the Java 11 virtual machine (JVM). Mitigating warm-up effects for benchmarks of small programs (execution time < 2 s) may need a large number of runs [55]. However, since our macrobenchmarks for the scalability evaluation **GS** and **A** had much longer execution times (up to 300 s), 10 extra runs before the actual measurements and explicit garbage collector calls between runs were sufficient for the stabilization of performance.

All measurements were executed on a high-performance server ($2 \times$ AMD EPYC 7551 32-core, 64-thread 2 GHz CPU, 512 GiB RAM) with a hard memory limit of 32 GiB, 16 GiB of which were assigned to the JVM heap to account for additional memory usage by IP and LP solvers. While the model generator is single threaded, parallel garbage collection of the JVM could take advantage of the 8 CPU cores (16 hardware threads) assigned to a measurement.

Results. The median running times of the approaches for different model sizes are shown in Fig. 10.

GS/O frequently ran out of the 300 s limit when generating models larger than 30 objects. Timeouts were less frequent in the case of 20 and 40 objects, which caused the median execution time of all runs (including timed out ones) to be discontinuous. This phenomenon can be partially explained by the interaction of type scope bounds and structural multiplicity constraints in **Sat**, which are somewhat easier to satisfy for these model sizes. **GS/S/TH** only reached the time limit for 80 and 90 objects. The median execution times for **GS/S/IP** and **GS/S/LP** were much smaller, not exceeding 65 s to generate models with 100 objects. This makes them the only approaches that were able to produce models of this size.

For all approaches, most of the execution time was spent in the decision and scope propagation, state coding, and exploration steps. The overhead of scope analysis remained below 3.1 s even for the largest generated models, which is negligible compared to other phases of model generation.

For models with 90 elements, **TH** scope analysis reduced number of states (partial models) explored during successful model generation from 41 000 (**GS/O**) to 36 000. **IP** and **LP** further reduced this to around 4000 states, indicating the effectiveness of scope analysis in discarding partial models with no valid concretization. While **IP** and **LP** reduced the state space virtually identically, linear programming (**LP**) was slightly faster: The overall runtime of the external solver was 1.7 s when generating 100-object models compared to the 3.1 s of **IP**.

RA1 Object scope analysis can significantly reduce both the execution time and the state space of model generation. Linear programming can provide the largest reductions with only a minor overhead of external solver calls.

4.3 Scalability of model generation

Setup. *This experiment aims at determining whether our model generation runs in practical time for test case generation with type scopes.* We generated models with increasing size in the **Sat**, **Sct**, and **Met** domains. For answering **RQ2.1**, we used scope bounds (**+S**) based on engineering expectations for **Sat**, and bounds based on real type distributions for **Sct** and **Met** (see the elaboration of **RQ4**). For answering **RQ2.2**, type scope bounds were omitted (**−S**) by definition.

The hardware environment and measurement protocol was identical to that of **RQ1.** We compared the scalability of the following model generators:

- **A**: Alloy Analyzer [12] v4.2 is a popular model finder based of SAT solving (we used the default Sat4J background solver). We translated the model generation problem into an Alloy model by known mappings [13]. We benchmarked both the Sat4J (**A/S4J**) and MiniSat (**A/MS**) background solvers.
- **GS/O:** To generate models with type scope bounds using the original *GraphSolver*, the bounds were translated into WF constraints.
- **GS/S:** Following the findings of **RQ1,** our graph generator used **LP** for object scope analysis.

Results. Figs. 4 and 5 show the execution times of the generators. The random exploration and backjumping heuristics caused large variance in the execution time, including frequent (but nondeterministic) timeouts of **GS/O** for larger models. To enable the in-depth analysis of these effects, the figures show boxplots of *successful* execution times for a given model size. Thus, the medians for **GS/O** are lower than those in Fig. 10, which were computed across all (successful or unsuccessful) runs. A red line chart shows

2. We also experimented with the νZ [54] v4.8.5 optimizing SMT solver. With *Real* variables (used as an LP solver) it produced results similar to CBC and CLP, albeit it performed object scope analysis slightly slower. With *Integer* variables (used as an IP solver), it produced out-of-memory errors. Results were omitted for space considerations.

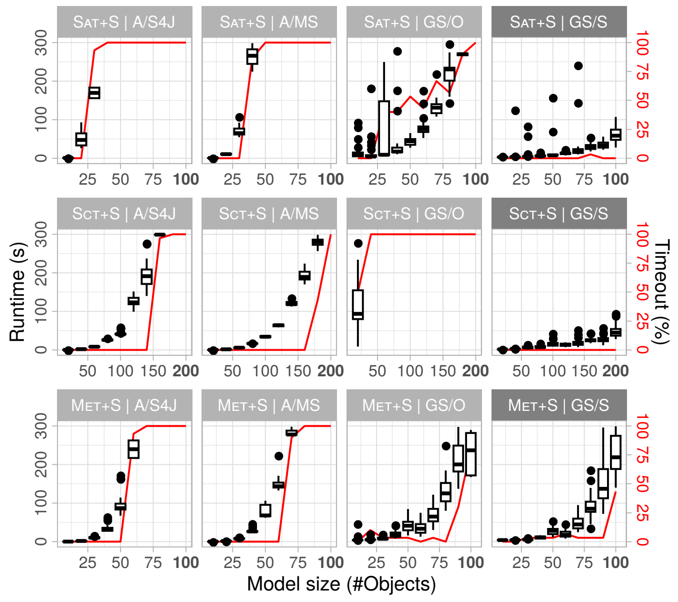


Fig. 4. Model generation with type scope bounds

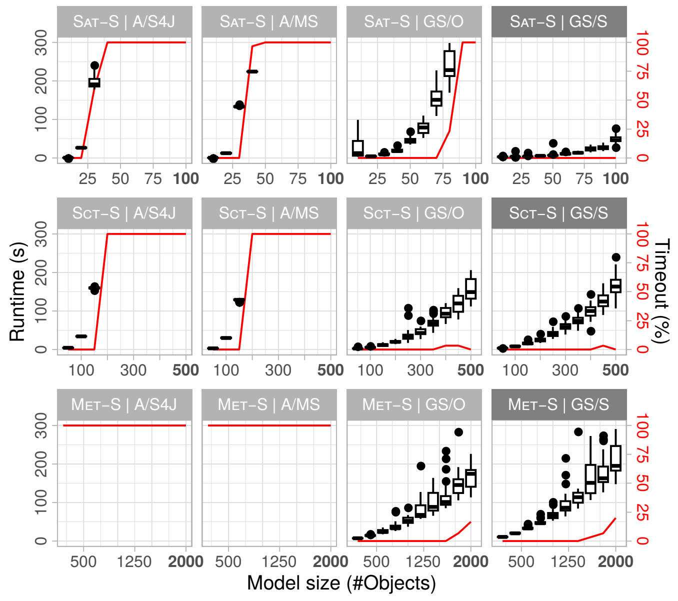


Fig. 5. Model generation without type scope bounds

the percentage of *unsuccessful* (timed out) executions out of the 30 runs for a given model size.

A encountered out of memory errors as the SAT problems grew too large with the increase of the desired number of objects in the models. In contrast, **GS/O** and **GS/S** were only limited by the execution timeout as limit (and hence the number of partial models the could explore) thanks to the concise representation of the space state by PMs.

It is clear that type scope bounds make the model generation tasks more challenging. For **SAT**, **GS/O** was unable to generate any model of 100 objects. With scope bounds, timeouts started to appear from 30 objects, while without bounds, models with up to 70 elements were generated without timing out. **GS/S** could generate models with 100 objects within 108 sec. However, **GS/S** with scope bounds exhibited some random slowdowns, where generation took an exceedingly long time or reached the time limit. These slowdowns, which were not experienced during model generation *without* scope bounds, could possibly be mitigated by refining the backjumping and restarting strategies.

The interaction of type scope bounds with structural and WF constraints in **SCT** made generation of models with realistic type distributions difficult. **GS/O** failed to generate any model of 40 objects or larger, while **GS/S** could generate models with 200 objects within 95 sec.

The removal of type scopes bounds greatly simplifies the task. Both **GS/O** and **GS/S** could produce models with up to 500 objects. In this domain, type scope analysis in **GS/S** yielded a median overhead of 16 s (on a total runtime of 149 s) without reducing the state space (and thus the execution time) of the generator compared to **GS/O** for models with 500 objects.

As **MET** does not contain any structural multiplicity constraints or WF constraints that affect the number of possible objects in the model, **GS/S** could only analyze the type scope bounds themselves. This reduced the median runtime of successful model generation by 18 s and the fraction of timed out runs by 36%. Like **SCT**, the removal of type scope bounds in **MET** made the problem easier. **GS/O** and **GS/S** could generate models with up to 2000 elements with similar performance (with a median scope analysis overhead of 21 s for models of 2000 objects). **A** failed to produce a model even for the smallest size (200 objects) in this scenario.

As a stress test, we also determined the maximum size of a model that **GS/S** can generate within the time limit of 5 minutes. With type scope bounds, these were 155 objects for **SAT**, 436 for **SCT**, and 121 for **MET**. Without satisfying the type scope bounds, much larger models are possible: 157 objects for **SAT**, 649 for **SCT**, and 2631 for **MET**.

RA2.1 For model generation problems with type scope bounds, object scope analysis improves the scalability model generation. The effect is most visible with up to 7-fold reduction in execution times when the type scope bounds interact with structural multiplicity constraints and WF constraints.
RA2.2 In model generation problems without type scope bounds, object scope analysis improves the scalability of model generation in domains with complex structural multiplicity constraints. When no such constraints are present, where is no performance improvement, but the overhead incurred by the analysis remains small.

4.4 Behavior on unsatisfiable problems

Setup. *The purpose of this experiment is to assess the performance degradation occurring in our approach in case of unsatisfiable problems.* Due to the lack of existing benchmark sets of unsatisfiable model generation problems, we introduces two modifications to the domains from **RQ2**.

Firstly, we extended each domain with a negated WF constraint, obtaining model generation problems with unsatisfiable WF constraints ($+\lightning_{\mathbf{WF}}$). For example, in **SAT**$+\lightning_{\mathbf{WF}}$, we added the error pattern $\varphi'_4(s) := \neg\varphi_4(s)$, which specifies that no Spacecraft may have a communication path to the GroundStation. Combined with the original φ_4 that forces such communication paths for all Spacecraft, the set of WF constraints $\{\varphi_1, \ldots, \varphi_4, \ldots \varphi_8, \varphi'_4\}$ have no consistent model. Error patterns for **SCT**$+\lightning_{\mathbf{WF}}$ and **MET**$+\lightning_{\mathbf{WF}}$ were defined analogously.

Secondly, we also studied the effect of *unsatisfiable type scope constraints* ($+\lightning_{\mathbf{S}}$), i.e., type scope constraints that correspond to no well-formed models. We changed the required number of objects such that multiplicity (*MUL*) and containment (*CON*) constraints cannot be satisfied due to type scope bounds, e.g., in **SAT**$+\lightning_{\mathbf{S}}$, we required at least 30% of the objects be Satellites but only 25% be CommSubsys instances, despite a Satellite having to contain at least one CommSubsys. We omitted **MET** from this benchmark, as it does not have any *MUL* constraints on *CON* relations.

Although we selected the type fractions to make type scope bounds unsatisfiable, rounding the fractions to whole numbers (quantization errors) of objects may cause the problems to be nevertheless satisfiable for very small instances. Thus, we had to account for these small satisfiable instances in our analysis.

Results. Fig. 6 shows the execution times of the generators on $+\lightning_{\mathbf{WF}}$ problems up to 15 objects.

Even though **GS/S** are primarily aimed at model generation, and thus had to explore a large portion of possible partial models before concluding unsatisfiability, they remained competitive in **SAT**$+\lightning_{\mathbf{WF}}$ and **SCT**$+\lightning_{\mathbf{WF}}$ in problems with up to 9 and 11 objects, respectively. Because **SAT** is unsatisfiable for less than 10 objects (even without $+\lightning_{\mathbf{WF}}$), **GS/S** could terminate without exploring the state space for the first 5 cases. In **SCT**$+\lightning_{\mathbf{WF}}$, although **GS/S** could not outright avoid state space exploration, it explored 16 times less states than **GS/O** thanks to scope analysis. **A**, which is much better suited for problems with unsatisfiable constraints, managed to prove unsatisfiability within 4 s for all model sizes in **SAT**$+\lightning_{\mathbf{WF}}$ and **SCT**$+\lightning_{\mathbf{WF}}$.

MET$+\lightning_{\mathbf{WF}}$ was more difficult for all approaches: while **A** could prove unsatisfiability with up to 11 objects (running out of memory at 12 objects), **GS/O** and **GS/S** did not terminate within the time limit even for 5 objects, exploring 19 000 states before timeout.

Figs. 7 and 8 show the execution times of the approaches on $+\lightning_{\mathbf{S}}$ problems. For small $+\lightning_{\mathbf{S}}$ problems with up to $n = 15$ objects, both **A** and **GS/S** terminated successfully within 5 s except in the cases of $n = 12$ and 13 in **SAT**$+\lightning_{\mathbf{S}}$ for **GS/S**. Due to the rounding of the type scope fractions into whole number bounds, these cases did not result in immediate unsatisfiable systems of linear equations. Therefore, **GS/S** had to explore 4982 and 5413 states, respectively, before concluding unsatisfiability. There also was a rounding effect that made **SCT**$+\lightning_{\mathbf{S}}$ satisfiable for $n = 8$. The model was found by **GS/S** after exploring 16 states. **GS/O** ran out of time after 10 objects, because it had to exhaustively enumerate partial models.

For larger problems with up to 100 objects for **SAT**$+\lightning_{\mathbf{WF}}$ and up to 200 objects for **SCT**$+\lightning_{\mathbf{WF}}$ in Fig. 8, the execution time of **GS/S** remained constant below 5 s. In contrast, the execution times of **A** increased cubically with n.

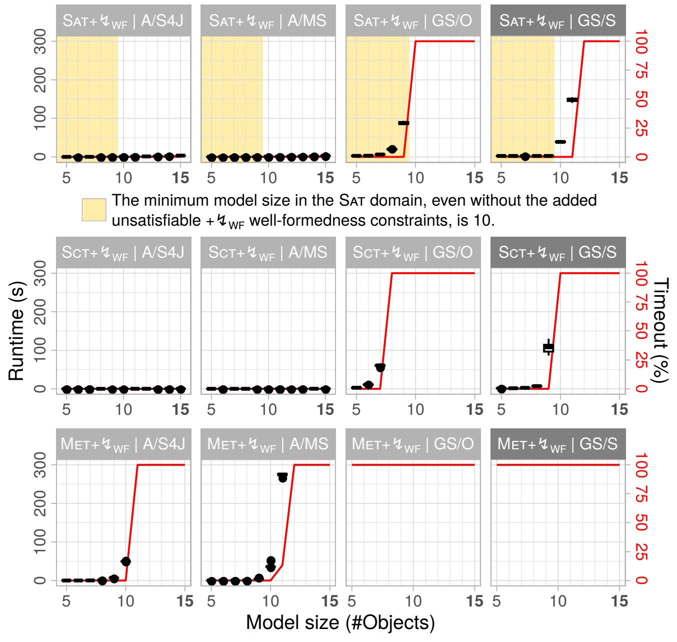


Fig. 6. Model generation with unsatisfiable well-formedness constraints with problem sizes up to 15 objects

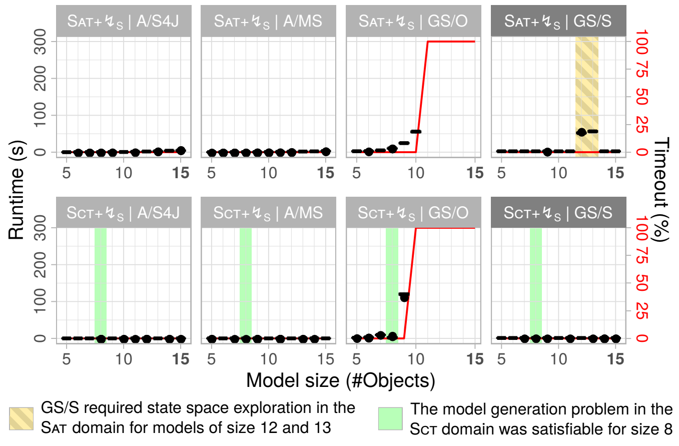


Fig. 7. Model generation with unsatisfiable type scope bounds with problem sizes up to 15 objects

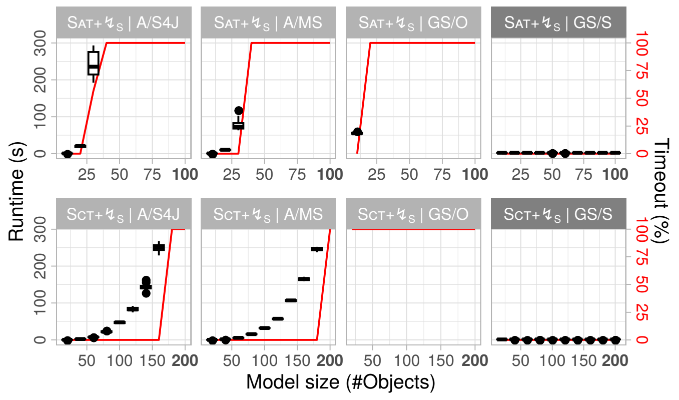


Fig. 8. Model generation with unsatisfiable type scope bounds with problem sizes up to 100 and 200 objects

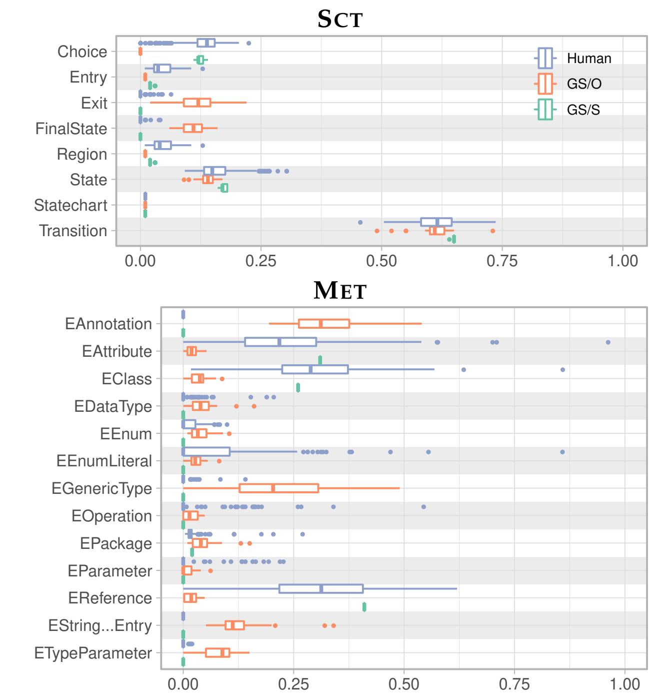


Fig. 9. Fractions of objects of given types in **SCT** and **MET**

RA3 For model generation problems with unsatisfiable well-formedness constraints, object scope analysis can improve the scalability of search space exploration for model generators. However, SAT solvers are better suited to tackle such problems. For model generation problems with unsatisfiable type scope bounds, object scope analysis can eliminate the need for exploring the state space, and the time taken for proving unsatisfiability is independent from the (potential) size of the state space.

4.5 Type distributions of models

Setup. *This experiment aims at comparing test generation approaches (without and with type scope bounds) where the test engineer desires to avoid unrealistic test models.* To address **RQ4**, we calculated the distribution of the fractions of objects of given types (i.e. the number of objects of a type in the model divided by the model size) of human (manually created) models, and compared them to the type distributions of automatically generated models. The use of type distributions as means of realistic nature of models was motivated by [18], [19]. As human models were only available for **SCT** and **MET**, we excluded **SAT** from this comparison.

- **Human:** We gathered 304 **SCT** models with sizes between 90 and 110 objects that were submitted as part of a homework assignment [56], where students solved similar (but not identical) modeling challenges. For **MET**, we collected 153 manually created class diagrams (those generated from XML schema were excluded) with sizes between 50 and 200 from open source projects hosted on GitHub[3].
- **GS/O:** For both domains, we generated 30 models of 100 objects (without any type scope bounds) with the original *GraphSolver* [32] tool.
- **GS/S:** We generated models with realistic type distributions with our graph solver enhanced with type scope support. To determine the lower and upper bounds for each type, we computed the lower $Q_i^{(1)}$ and upper $Q_i^{(3)}$ quartiles of the object fractions in the **Human** models for each non-abstract class C_i. Then for the generation of models with $n = 100$ objects, we added the type scope bounds $\lfloor Q_i^{(1)} n \rfloor \leq \mathsf{C}_i \leq \lceil Q_i^{(3)} n \rceil$.

Results. The distribution of type fractions is shown in Fig. 9. In **SCT**, **GS/O** generated a large number of Exit and FinalState objects compared to the **Human** while it almost entirely omitted Choice, Entry, and Region. The average discrepancy between the type distribution of **Human** models in **GS/O** models is 25 objects per model (25%) that would need a different type to match the **Human** distribution.

In **MET**, **GS/O** overused the EAnnotation, EGenericType, EStringToStringMapEntry, and ETypeParameter classes at the expense of EAttribute, EClass, and EReference objects. The average discrepancy was 83 objects per model (with 100 objects). Models generated by **GS/S** had identical type distributions, which for EAttribute and EClass coincided with the upper type scope bound.

Therefore, **GS/O** failed to generate models matching the type distributions of **Human** models. In contrast, **GS/S** can be parameterized to satisfy type distribution requirements, e.g., *probabilistic types* and *histograms* [19]. Furthermore, to capture more complex correlation between distributions of different types, users can inspect generated models and easily (albeit manually) refine type scope bounds to exclude results that are not realistic, using an iterative process based on previously generated undesired models.

All models generated by **GS/O** and **GS/S** were connected (i.e. no islands or forest of nodes) and they were structurally different from each other, which is guaranteed by the underlying state space exploration strategy [32].

RA4 Models generated without type scopes bounds greatly differ in type distribution compared to human (manually created) models. The use of type scope bounds allows generating nontrivial, connected graph models with designated type distributions.

While type distributions were found to be a useful metric to characterize the realistic nature of models [8], [18], further investigations are necessitated along various metrics to claim that the auto-generated models are truly realistic.

We also confirmed that the internal diversity [20] of the synthesized models is not impacted negatively by the proposed approach. The relevance of this metric in mutation testing is shown in [20].

4.6 Limitations and threats to validity

Limitations. Our approach shares some of its strengths and limitations with *GraphSolver* [32]. Namely, it operates over connected sparse graphs with edges and relations, i.e., without edge identities or parallel edges (which is suitable to represent standard EMF models).

3. We queried the GitHub (https://github.com) API for the 1000 most recent Ecore models as of July 31st, 2019 and filtered for model size and the lack of XML schema.

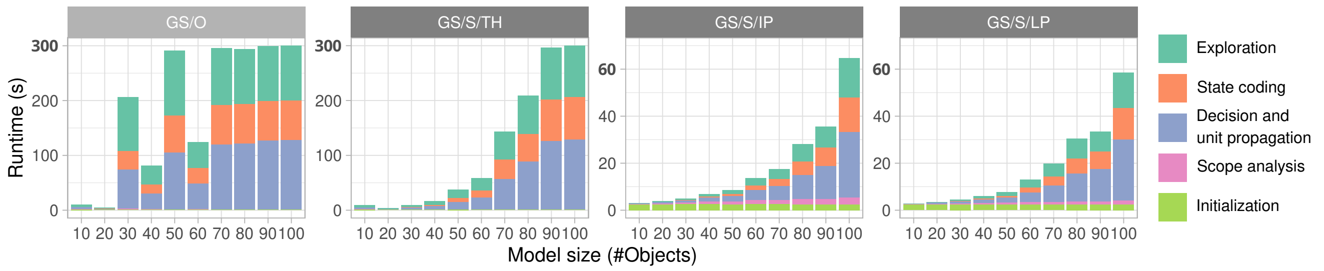


Fig. 10. Comparison of scope analysis methods on the **SAT** domain

The expressive power of the graph predicates capturing WF constraints is equivalent to first-order logic with transitive closure over binary predicates. While type scope bounds and object scopes do not bring additional expressiveness, so they can be transformed back into WF constraints, they considerably improve scalability in various domains. Object scopes consisting of linear inequalities can exactly encode type scopes bounds (including the bound on the overall model size), and they can also encode weakened versions of structural *MUL* and *CON* constraints (including the XOR between different containment relations of objects), guiding state space exploration in challenges that often arise from class diagrams. However, for model generation tasks without such constraints, it may not be possible to (even manually) encode useful linear inequalities, and the introduced object scope analysis may pose a slight overhead over the baseline generator.

The sound and complete set of decision rules allow formal reasoning within the bounded scope defined by type scope bounds. However, unlike many SAT and SMT solvers, there is currently no support for an unsatisfiable core (a minimal contradictory set of formulas) that would highlight the contradiction between WF constraints or type scopes.

The work presented in this paper only considers classes and references, but not attributes. While the three-valued logic framework can support basic attributes, placing and maintaining scope bounds for attribute values would require additional abstractions, such as [40].

In unsatisfiable problems, proving unsatisfiability with a model generator may require exponential time to exhaustively traverse the search space if the search cannot be aborted early with scope analysis. Thus such problems may be more amenable to SAT solving instead.

The generation of models with realistic type distributions assumes the availability of real models to determine type histograms. For ensuring realistic properties other than type distributions, additional heuristics may be needed.

Internal validity. Our scalability experiments incorporated a warm-up phase prior to actual measurements and garbage collector calls between actual measurements to reduce variance of execution times due to the JVM (but not due to the inherent behavior of the model generators). To further mitigate disturbances from the environment, each measurement was pinned to a single memory controller and the associated CPU cores on our server. We used default configurations for the external **IP** and **LP** solvers, as well as **A**. Domain-specific fine-tunings may reduce the execution times of these programs, but in most cases they were already negligible.

As noted in Section 2.1, **A** only supports limited type scopes. The $+\mathbf{S}$ problems cannot be formalized in **A** without the use of the # operator, even if lower bound constraints are omitted. However, as **A** performance was similar on both $+\mathbf{S}$ and $-\mathbf{S}$ problems, our encoding of the bounds likely did not introduce scalability bottlenecks.

For determining realistic type distributions of the industrial modeling languages, we considered manually constructed and automatically generated models of similar size to minimize discrepancies caused by different scales. For **SAT**, the distribution were prescribed manually. The behavior of the model generators did not change drastically upon changing the prescribed distribution, as long as the arising type scope bounds remained satisfiable.

External validity. Our measurements cover 3 domains (1 from a design space exploration challenge published by NASA researchers, 2 from industrial modeling languages) both with and without *realistic type scope bounds.* All domains had *complex structural* and *WF constraints* that interacted in various ways with the type scope bounds. Consequently, our experimental scalability results of our graph generator are likely generalizable to other domains of similar size.

As the performance of object scope analysis based on IP and LP depends on the selected external solver, we integrated the νZ optimizing SMT solver in addition to the well-known solvers from the COIN-OR project. In case of LP problems, performance was comparable to CLP, while for IP problems, CBC proved to be significantly better. Therefore, the reported scalability of IP and LP object scope propagation likely matches what is achievable with state-of-the-art external solvers.

5 RELATED WORK

Logic Solver Approaches. Several approaches map a model generation problem into a logic problem, which is solved by underlying SAT/SMT-solvers. Complete frameworks with standalone specification languages include Formula [14] (using the Z3 SMT-solver [24]), Alloy [12] (using SAT-solvers like Sat4j [22]) and Clafer [57] (using reasoners like Alloy).

There are several approaches aiming to validate standardized engineering models enriched with OCL constraints [58] by relying upon different back-end logic-based approaches such as constraint logic programming [36], [59], [60], SAT-based model finders (like Alloy) [7], [16], [35], [50], [51], [61], [62], [63], CSP solvers [64], first-order logic [65], constructive query containment [66] or higher-order logic [67]. Partial snapshots and WF constraints can be uniformly represented as constraints [7]. Growing models are supported in [50], [68] for a limited set of constraints.

Scalability of all these approaches are limited to small models / counter-examples. Furthermore, these approaches are either a priori bounded (where the search space needs to be restricted explicitly) or they have decidability issues. As our approach is independent from the actual mapping of constraints to logic formulae, it could potentially be integrated with most of the above techniques by complementing or replacing the back-end solvers.

Uncertain Models. Partial models are similar to uncertain models, which offer a rich specification language [30], [69] amenable to analysis. They a more user-friendly language compared to 3-valued interpretations, but without handling additional WF constraints. Potential concrete models compliant with an uncertain model can be synthesized by the Alloy Analyzer [31], or refined by graph transformation rules [70]. Each concrete model is derived in a single step, thus their approach is not iterative like ours. Scalability analysis is omitted from these papers, but refinement of uncertain models is always decidable.

Approaches like [71] analyze possible matches and executions of model transformation rules on partial models by using a SAT solver (MathSAT4) or by automated graph approximation (referred to as "lifting"), or by graph query engines with [26]. As a key difference, our approach carries out model refinement while simultaneously evaluating graph query evaluation.

Iterative Approaches. Iterative approaches generate models by multiple solver calls. In [50] models are generated in by calling Alloy in multiple steps, where each step extends the instance model by a few elements. This approach scaled up to 50 object in 45 s for generating valid Yakindu Statecharts. An iterative approach is proposed *specifically for allocation problems* in [72] based on Formula. An iterative, counter-example guided synthesis is proposed for higher-order logic formulae in [73], but the size of models is fixed and smaller than 50 objects.

Symbolic Model Generation Techniques. Certain techniques use abstract (or symbolic) graphs for analysis purposes. A tableau-based reasoning method is proposed for graph properties [74], [75], [76], which automatically refines solutions based on well-formedness constraints, and handles the 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 [74], [75] resolves the graph constraints to a concrete solution. Therefore our approach is able to exploit efficient graph query engines to evaluate partial solutions, while those techniques are demonstrated on small (< 10 objects) graphs or with no scalability evaluation.

Different approaches use abstract interpretation [77], or predicate abstraction [39], [40], [78] for partial modeling. In those approaches, concretization is used to materialize (typically small) counter-examples for designated safety properties in a graph transformation system. However, their focus is to support model checking of abstract graph transformation systems, which can evaluate complex trajectories, but do not scale in the size of the models.

Additionally, counter abstractions by *Petri graphs* were used in the verification of graph transformation systems [79] and as heuristic functions for rule-based design-space exploration [80]. The Augur framework [41], [42], [81] uses similar counter abstraction on graph properties for in graph transformation systems, which can be analyzed as a transition system. As a key difference, a graph transformation rule can both increase or decrease amounts in abstract graphs, while in our approach the constraints are respecting the refinement relation, thus we can utilize IP and LP solvers instead of model checkers.

Smart bound selection for the number of objects was used in the satisfiability checking of OCL formulas in [82].

Numerical Abstractions. Verification of programs containing numerical (integer or real) variables by abstract interpretation relies of numerical abstract domains [83], [84], including polyhedra defined by systems of linear equations [85], [86], as a key component to over- and under-approximate the sets of possible program states. Numerical abstract domains are combined with graph abstractions in two main ways to verify heap and pointer based programs.

Firstly, numerical abstract domains may summarize object attributes (field) in value analysis of heap programs [40], [87], [88]. Summarized dimensions [78] were introduced to succinctly represent attributes of a potentially unbounded set of objects via multi-objects. This approach can be seen as complementary to ours, as it enables attribute handling in three-valued partial models, and allows checking for refinements by abstract subsumption [89].

Secondly, numerical abstract domains can aid reasoning about the number of objects in a graph (usually a program heap) by structural counter abstraction [90]. This approach is closely related to ours, but its use is limited to program verification. In contrast, we explicitly incorporate uncertain types and references by three-valued partial modeling to enable model generation.

Model-based quantifier instantiation approach [91] in SMT solvers for finite model finding also relies on counter abstractions. It can be seen as a dual to scoped model generation: it aims to merge terms to satisfy finiteness constrains instead of splitting multi-objects to add new objects.

6 Conclusions and future work

In this paper, we propose a new 3-valued scoped partial modeling formalism which allows to explicitly represent multiplicity constraints on the size of partial models with a linear inequality system. Those constraints cover the requested size of the completed model for each class (type) and the additional constraints imposed by the metamodel (e.g., reference multiplicities) and well-formedness constraints. The resultant multiplicity constraint can be efficiently solved by an underlying IP or LP solver to get a more precise view on the number of objects in a potential concretization of the partial model, or to detect infeasible numerical requirements on it. Based on the advanced numerical reasoning on partial models, we extend the graph solver algorithm of [21], [32] with scoped partial models and numerical reasoning using IP or LP solvers, which greatly improves the performance of the solver (and outperforms related solvers like [13]). Additionally, the proposed technique enables the efficient use of type scopes, which allows the generation of more realistic or useful models.

As future work, we plan to extend the model generator with numerical *optimization* with respect to a user-defined cost function (or goal function), thus allowing the generation of graph models with optimal properties (like smallest models or cheapest models with respect to a cost-function).

REFERENCES

[1] RTCA, Inc., "DO-330 sofware tool qualification considerations," 2011. [Online]. Available: https://standards.globalspec.com/std/1461615/rtca-do-330

[2] A. Milicevic, S. Misailovic, D. Marinov, and S. Khurshid, "Korat: A tool for generating structurally complex test inputs," in *ICSE*, 2007, pp. 771–774.

[3] S. Khurshid and D. Marinov, "TestEra: Specification-based testing of Java programs using SAT," *Autom. Softw. Eng.*, vol. 11, no. 4, pp. 403–434, 2004.

[4] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *KES-AMSTA*, ser. LNCS, vol. 7327. Springer, 2012, pp. 504–513.

[5] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *ICSE*, 2018, pp. 1016–1026.

[6] M. Z. Iqbal, A. Arcuri, and L. Briand, "Environment modeling and simulation for automated testing of soft real-time embedded software," *Softw. Syst. Model.*, vol. 14, no. 1, pp. 483–524, 2015.

[7] O. Semeráth, Á. Barta, Á. Horváth, Z. Szatmári, and D. Varró, "Formal validation of domain-specific languages with derived features and well-formedness constraints," *Softw. Syst. Model*, vol. 16, no. 2, pp. 357–392, 2017.

[8] D. Varró, O. Semeráth, G. Szárnyas, and Á. Horváth, "Towards the automated generation of consistent, diverse, scalable and realistic graph models," in *Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig*, 2018, pp. 285–312.

[9] The Object Management Group, "Object Constraint Language, v2.4," 2014. [Online]. Available: https://www.omg.org/spec/OCL/2.4

[10] D. Varró and A. Balogh, "The model transformation language of the VIATRA2 framework," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 214–234, 2007.

[11] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró, "EMF-IncQuery: An integrated development environment for live model queries," *Sci. Comput. Program.*, vol. 98, pp. 80–99, 2015.

[12] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Tran. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, 2002.

[13] E. Torlak and D. Jackson, "Kodkod: A relational model finder," in *TACAS*. Springer, 2007, pp. 632–647.

[14] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Reasoning about metamodeling with formal specifications and automatic proofs," in *MODELS*. Springer, 2011, pp. 653–667.

[15] E. K. Jackson and J. Sztipanovits, "Towards a formal foundation for domain specific modeling languages," in *EMSOFT*. ACM, 2006, pp. 53–62.

[16] M. Kuhlmann, L. Hamann, and M. Gogolla, "Extensive validation of OCL models by integrating SAT solving into USE," in *TOOLS*, ser. LNCS, vol. 6705, 2011, pp. 290–306.

[17] J. Cabot, R. Clarisó, and D. Riera, "On the verification of UML/OCL class diagrams using constraint programming," *J. Syst. Softw.*, vol. 93, pp. 1–23, 2014.

[18] G. Soltana, M. Sabetzadeh, and L. C. Briand, "Practical model-driven data generation for system testing," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1902.00397

[19] G. Soltana, M. Sabetzadeh, and L. C. Briand, "Synthetic data generation for statistical testing," in *ASE*, 2017, pp. 872–882.

[20] O. Semeráth, R. Farkas, G. Bergmann, and D. Varró, "Diversity of graph models and graph generators in mutation testing," *Int. J. Softw. Tools Technol. Transf.*, 2019.

[21] O. Semeráth, A. A. Babikian, S. Pilarski, and D. Varró, "VIATRA Solver: A framework for the automated generation of consistent domain-specific models," in *ICSE Demo*. IEEE, 2019, pp. 43–46.

[22] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *J. Satisfiability Boolean Model. Comput.*, vol. 7, pp. 59–64, 2010.

[23] N. Eén and N. Sörensson, "An extensible SAT-solver," in *ICTAST*. Springer, 2003, pp. 502–518.

[24] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*. Springer, 2008, pp. 337–340.

[25] E. K. Jackson, S. Gabor, and J. Sztipanovits, "Diversely enumerating system-level architectures," Tech. Rep. MSR-TR-2013-56, 2013.

[26] O. Semeráth and D. Varró, "Graph constraint evaluation over partial models by constraint rewriting," in *ICMT*, 2017, pp. 138–154.

[27] M. Al-Refai, W. Cazzola, and S. Ghosh, "A fuzzy logic based approach for model-based regression test selection," in *MODELS*. IEEE, 2017, pp. 55–62.

[28] S. Edunov, D. Logothetis, C. Wang, A. Ching, and M. Kabiljo, "Generating synthetic social graphs with darwini," in *ICDCS*. IEEE, 2018, pp. 567–577.

[29] D. Honfi and Z. Micskei, "Classifying generated white-box tests: an exploratory study," *Softw. Qual. J.*, vol. 27, pp. 1339–1380, 2019.

[30] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *ICSE*. IEEE, 2012, pp. 573–583.

[31] R. Salay, M. Famelis, and M. Chechik, "Language independent refinement using partial modeling," in *FASE*, ser. LNCS, J. de Lara and A. Zisman, Eds. Springer, 2012, vol. 7212, pp. 224–239.

[32] O. Semeráth, A. S. Nagy, and D. Varró, "A graph solver for the automated generation of consistent domain-specific models," in *ICSE*. ACM, 2018 2018.

[33] S. J. I. Herzig, S. Mandutianu, H. Kim, S. Hernandez, and T. Imken, "Model-transformation-based computational design synthesis for mission architecture optimization," in *IEEE Aerospace Conf.* IEEE, 2017.

[34] *Eclipse Modeling Framework*, The Eclipse Project, 2017, http://www.eclipse.org/emf.

[35] S. M. A. Shah, K. Anastasakis, and B. Bordbar, "From UML to Alloy and back again," in *MoDeVVa*. ACM, 2009.

[36] J. Cabot, R. Clarisó, and D. Riera, "UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming," in *ASE*. ACM, 2007, pp. 547–548.

[37] U. Nickel, J. Niere, and A. Zündorf, "The FUJABA environment," in *ICSE*, 2000, pp. 742–745.

[38] A. Rensink, I. Boneva, H. Kastenberg, and T. S. en, "User manual for the GROOVE tool set," 2012. [Online]. Available: https://groove.ewi.utwente.nl/wp-content/uploads/usermanual1.pdf

[39] T. W. Reps, M. Sagiv, and R. Wilhelm, "Static program analysis via 3-valued logic," in *CAV*, 2004, pp. 15–30.

[40] P. Ferrara, R. Fuchs, and U. Juhasz, "TVAL+: TVLA and value analyses together," in *SEFM*, ser. LNCS, vol. 7504. Springer, 2012, pp. 63–77.

[41] P. Baldan, A. Corradini, and B. König, "A static analysis technique for graph transformation systems," in *CONCUR*, ser. LNCS, vol. 2154. Springer, 2001, pp. 381–295.

[42] B. König and V. Kozioura, "Augur 2 – a new version of a tool for the analysis of graph transformation systems," *Electr. Notes Theor. Comput. Sci.*, vol. 211, pp. 201–210, 2008, gT-VMT 2006.

[43] D. Varró, G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, and Z. Ujhelyi, "Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework," *Softw. Syst. Model.*, vol. 15, no. 3, pp. 609–629, 2016.

[44] G. Szárnyas, B. Izsó, D. Harmath, G. Bergmann, and D. Varró, "IncQuery-D: A distributed incremental model query framework in the cloud," in *MODELS*, ser. LNCS, vol. 8767. Springer, 2014, pp. 653–669.

[45] A. Benelallam, A. Gómez, M. Tisi, and J. Cabot, "Distributed model-to-model transformation with ATL on MapReduce," in *SLE*. ACM, 2015, pp. 37–48.

[46] H. Abdeen, D. Varró, H. Saharoui, A. S. Nagy, Á. Hegedüs, and Á. Horváth, "Multi-objective optimization in rule-based design-space exploration," in *ASE*. ACM, 2014, pp. 289–300.

[47] A. Rensink, "Isomorphism checking in GROOVE," ser. LNCS, vol. 4549. Springer, 2006.

[48] F. Yu, T. Bultan, and E. Peterson, "Automated size analysis for OCL," in *ESEC / FSE*. ACM, 2007, p. 331–340.

[49] Yakindu Statechart Tools, *Yakindu*, http://statecharts.org/.

[50] O. Semeráth, A. Vörös, and D. Varró, "Iterative and Incremental Model Generation by Logic Solvers," in *FASE*, 2016, pp. 87–103.

[51] F. Büttner, M. Egea, J. Cabot, and M. Gogolla, "Verification of ATL transformations using transformation models and model finders," in *ICFEM*. Springer, 2012, pp. 198–213.

[52] H. Wu, "An SMT-based approach for generating coverage oriented metamodel instances," *Int. J. Inf. Syst. Model. Design*, vol. 7, no. 3, 2016.
[53] B. Alkhazi, C. Abid, M. Kessentini, D. Leroy, and M. Wimmer, "Multi-criteria test cases selection for model transformations," *Autom. Softw. Eng.*, 2020.
[54] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ – an optimizing SMT solver," in *TACAS*, ser. LNCS, vol. 9035. Springer, 2015, pp. 194–199.
[55] E. Barrett, C. F. Bolz-Tereick, R. Killick, S. Mount, and L. Tratt, "Virtual machine warmup blows hot and cold," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, 2012, article No.: 52.
[56] *System Modeling*, Budapest Univ. of Technology and Economics, https://portal.vik.bme.hu/kepzes/targyak/VIMIAA00/en/.
[57] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski, "Clafer: unifying class and feature modeling," *Softw. Syst. Model.*, pp. 1–35, 2013.
[58] M. Gogolla, J. Bohling, and M. Richters, "Validating UML and OCL models in USE by automatic snapshot generation," *Softw. Syst. Model.*, vol. 4, pp. 386–398, 2005.
[59] J. Cabot, R. Clariso, and D. Riera, "Verification of UML/OCL class diagrams using constraint programming," in *ICSTW*, 2008, pp. 73–80.
[60] F. Büttner and J. Cabot, "Lightweight string reasoning for OCL," in *ECMFA*, ser. LNCS, vol. 7349. Springer, 2012, pp. 244–258.
[61] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "On challenges of model transformation from UML to Alloy," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 69–86, 2010.
[62] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, "Verifying UML/OCL models using boolean satisfiability," in *DATE*. IEEE, 2010, pp. 1341–1344.
[63] B. Meng, A. Reynolds, C. Tinelli, and C. Barrett, "Relational constraint solving in SMT," in *CADE*, ser. LNCS, vol. 10395. Springer, 2017, pp. 148–165.
[64] C. A. González, F. Büttner, R. Clarisó, and J. Cabot, "EMFtoCSP: a tool for the lightweight verification of EMF models," in *FormSERA*, 2012, pp. 44–50.
[65] B. Beckert, U. Keller, and P. H. Schmitt, "Translating the Object Constraint Language into First-order Predicate Logic," in *Proc. VERIFY, Workshop at FLoC*, 2002.
[66] A. Queralt, A. Artale, D. Calvanese, and E. Teniente, "OCL-Lite: Finite reasoning on UML/OCL conceptual schemas," *Data Knowl. Eng.*, vol. 73, pp. 1–22, 2012.
[67] H. Grönniger, J. O. Ringert, and B. Rumpe, "System model-based definition of modeling language semantics," in *FORTE*, ser. LNCS, vol. 5522. Springer, 2009, pp. 152–166.
[68] E. K. Jackson and J. Sztipanovits, "Constructive techniques for meta-and model-level reasoning," in *MODELS*. Springer, 2007, pp. 405–419.
[69] R. Salay and M. Chechik, "A generalized formal framework for partial modeling," in *FASE*, ser. LNCS. Springer Berlin Heidelberg, 2015, vol. 9033, pp. 133–148.
[70] R. Salay, M. Chechik, M. Famelis, and J. Gorzny, "A methodology for verifying refinements of partial models," *Journal of Object Technology*, vol. 14, no. 3, pp. 3:1–31, 2015.
[71] M. Famelis, R. Salay, A. Di Sandro, and M. Chechik, "Transformation of models containing uncertainty," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2013, pp. 673–689.
[72] E. Kang, E. Jackson, and W. Schulte, "An approach for effective design space exploration," in *Monterey Workshop*, ser. LNCS. Springer, 2010, vol. 6662, pp. 33–54.
[73] A. Milicevic, J. P. Near, E. Kang, and D. Jackson, "Alloy*: A general-purpose higher-order relational constraint solver," in *ICSE*, 2015, pp. 609–619.
[74] S. Schneider, L. Lambers, and F. Orejas, "Symbolic model generation for graph properties," in *FASE*, ser. LNCS, vol. 10202. Springer, 2017, pp. 226–243.
[75] K.-H. Pennemann, "Resolution-like theorem proving for high-level conditions," in *ICGT*, ser. LNCS, vol. 5214. Springer, 2008, pp. 289–304.
[76] A. S. Al-Sibahi, A. S. Dimovski, and A. Wasowski, "Symbolic execution of high-level transformations," in *SLE*. Springer, 2016, pp. 207–220.
[77] A. Rensink and D. Distefano, "Abstract graph transformation," *Electr. Notes Theor. Comput. Sci.*, vol. 157, no. 1, pp. 39–59, 2006.
[78] D. Gopan, F. DiMaio, N. Dor, T. Reps, and M. Sagiv, "Numeric domains with summarized dimensions," in *TACAS*, ser. LNCS, vol. 2988. Springer, 2004, pp. 512–529.
[79] D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer, "Termination analysis of model transformations by Petri nets," in *ICGT*, ser. LNCS, vol. 4178. Springer, 2006, pp. 260–274.
[80] Á. Hegedüs, Á. Horváth, and D. Varró, "A model-driven framework for guided design space exploration," *Autom. Softw. Eng.*, vol. 22, no. 3, pp. 399–436, 2015.
[81] B. König and V. Kozioura, "Counterexample-guided abstraction refinement for the analysis of graph transformation systems," in *TACAS*, ser. LNCS, vol. 3920. Springer, 2006, pp. 197–211.
[82] R. Clarisó, C. A. González, and J. Cabot, "Smart bound selection for the verification of UML/OCL class diagrams," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 412–426, 2019.
[83] A. Miné, "Weakly relational numerical abstract domains," Ph.D. dissertation, 2004. [Online]. Available: https://www-apr.lip6.fr/~mine/these/these-color.pdf
[84] G. Singh, M. Püschel, and M. Vechev, "A practical construction for decomposing numerical abstract domains," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, 2018, article no. 2.
[85] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *POPL*. ACM, 1978, pp. 84–96.
[86] R. Bagnara, P. M. Hill, and E. Zaffanella, "The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems," *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 3–21, 2008.
[87] S. Magill, J. Berdine, E. Clarke, and B. Cook, "Arithmetic strengthening for shape analysis," in *SAS*, ser. LNCS, vol. 4634. Springer, 2007, pp. 419–436.
[88] B. McCloskey, T. Reps, and M. Sagiv, "Statically inferring complex heap, array, and numeric invariants," in *SAS*, ser. LNCS, vol. 6337. Springer, 2010, pp. 71–99.
[89] S. Anand, C. S. Păsăreanu, and W. Visser, "Symbolic execution with abstraction," *Int. J. Softw. Tools Technol. Transf.*, vol. 11, no. 1, pp. 53–67, 2009.
[90] K. Bansal, E. Koskinen, T. Wies, and D. Zufferey, "Structural counter abstraction," in *TACAS 2013*, ser. LNCS, vol. 7795. Springer, 2013, pp. 62–77.
[91] A. Reynolds, C. Tinelli, A. Doel, and S. Kristić, "Finite model finding in SMT," in *CAV*, ser. LNCS, vol. 8044. Springer, 2013, pp. 640–655.

Kristóf Marussy is a PhD student at the Department of Measurement and Information Systems at Budapest University of Technology and Economics. He is also a research assistant at the MTA Lendület Cyber-Physical Systems Research Group. His research interest include the modeling and analysis of extra-functional properties of cyber-physical systems, and the synthesis of reliable architectures. He participated in research visits at the University of L'Aquila and McGill University.

Oszkár Semeráth is a research fellow at Budapest University of Technology and Economics and MTA Lendület Cyber-Physical Systems Research Group. His research focuses on modeling tools, logic solvers and graph generation, he is the main developer of the VIATRA Solver graph generator framework. His results were published in a book chapter, 3 journal papers with impact factor, in 12 conference papers, and won IEEE/ACM best paper award at the MODELS 2013 conference.

Dániel Varró is a full professor of software engineering at McGill University and at Budapest University of Technology and Economics. He is also a research chair of the MTA Lendület Cyber-Physical Systems Research Group. He is a co-author of more than 150 scientific papers with seven Distinguished Paper Awards, and two Most Influential Paper Awards. He regularly serves on the program committee of various international conferences in the field and serves on the editorial board of the Software and Systems Modeling journal (Springer) and Journal of Object Technology. He was a program committee co-chair of FASE 2013, ICMT 2014 and SLE 2016 conferences. He delivered a keynote talk at the IEEE CSMR 2012 and the SOFSEM 2016 conferences and at various international workshops and at the DSM-TP international summer school. He is a co-founder of the VIATRA model query and transformation framework, and IncQuery Labs Ltd., a technology-intensive Hungarian company.

A Proofs of Theorems 3 and 4

A.1 Proof of Theorem 3

First we will prove the statement

$$\mathcal{S}_P \vDash \#_v^{1/2}[\![\varphi]\!]^P_{abs \circ Z} < L \implies \mathcal{S}_Q \vDash \#_v^{1/2}[\![\varphi]\!]^Q_Z < L.$$

Consider the sets

$$A = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^P_{abs\circ Z, v\mapsto abs(x)} \geq 1/2\},$$
$$B = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^Q_{Z, v\mapsto x} \geq 1/2\},$$
$$C = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^P_{abs\circ Z, v\mapsto abs(x)} \geq 1/2, [\![\varphi]\!]^Q_{Z, v\mapsto x} = 0\}.$$

The set A contains the objects $x \in \mathcal{O}_Q$ that refine an $y = abs(x) \in \mathcal{O}_P$ for which φ was possibly true in P. The elements of B may satisfy φ in Q, while elements of C may have satisfies φ in P, but no longer satisfy it in Q. By Theorem 2 $[\![\varphi]\!]^Q_{v\mapsto x} \geq 1/2$ implies $[\![\varphi]\!]^P_{v\mapsto abs(x)} \geq 1/2$. Thus $B \subseteq A$ and $A = B \cup C$.

We replace the occurrences of each variable $\widehat{y} \in \mathcal{V}_P$ with $\sum\{\widehat{x} \mid abs(x) = y\}$ in $\mathcal{S}_P \vDash \#_v^{1/2}[\![\varphi]\!]^P_{abs\circ Z} < L$ to obtain

$$\mathcal{S}_P^{abs} \vDash \sum\{\widehat{x} \mid x \in A\} = \sum\{\widehat{x} \mid x \in B\} + \sum\{\widehat{x} \mid x \in C\} < L.$$

We have $\mathcal{S}_P^{abs} \vDash \sum\{\widehat{x} \mid x \in C\} \geq 0$, because each variable $\widehat{x}$ must take a nonnegative value in any solution of $\mathcal{S}_P^{abs}$. Moreover, because $\mathcal{S}_P \succcurlyeq_{abs} \mathcal{S}_Q$, we have $\mathcal{S}_Q \vDash \mathcal{S}_P^{abs}$ according to Definition 9. Therefore,

$$\mathcal{S}_Q \vDash \mathcal{S}_P^{abs} \vDash \sum\{\widehat{x} \mid x \in A\} < \sum\{\widehat{x} \mid x \in B\} < L.$$

Noting that $\#_v^{1/2}[\![\varphi]\!]^Q_Z = \sum\{\widehat{x} \mid x \in B\}$, we complete the proof of the first part by simplifying to $\mathcal{S}_Q \vDash \#_v^{1/2}[\![\varphi]\!]^Q_Z < L$.

In order to tackle

$$\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]^P_{abs \circ Z} > U \implies \mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]^Q_Z > U,$$

we consider the sets

$$A' = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^Q_{Z, v\mapsto x} = 1\},$$
$$B' = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^P_{abs\circ Z, v\mapsto abs(x)} = 1\},$$
$$C' = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^P_{abs\circ Z, v\mapsto abs(x)} \leq 1/2, [\![\varphi]\!]^Q_{Z, v\mapsto x} = 1\}.$$

Elements of A' surely satisfy φ. Elements of B' are refinements of objects in P that surely satisfied φ, while elements of C' only surely satisfy it in Q, but not in P. By Theorem 1, $[\![\varphi]\!]^P_{abs\circ Z, v\mapsto abs(x)} = 1$ implies $[\![\varphi]\!]^Q_{Z, v\mapsto x} = 1$. Thus $B' \subseteq A'$ and $A' = B' \cup C'$.

After substituting the variables in $\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]^P_{abs\circ Z} > U$ as before, we obtain $\mathcal{S}_P^{abs} \vDash \sum\{\widehat{x} \mid x \in B'\} > U$.

Again, we have $\mathcal{S}_P^{abs} \vDash \sum\{\widehat{x} \mid x \in C'\} \geq 0$, as well as $\mathcal{S}_Q \vDash \mathcal{S}_P^{abs}$. Therefore,

$$\mathcal{S}_Q \vDash \mathcal{S}_P^{abs} \vDash U < \sum\{\widehat{x} \mid x \in B'\} \leq \sum\{\widehat{x} \mid x \in B'\} + \sum\{\widehat{x} \mid x \in C'\} = \sum\{\widehat{x} \mid x \in A'\}.$$

Noting that $\#_v^{1}[\![\varphi]\!]^Q_Z = \sum\{\widehat{x} \mid x \in A'\}$, we simplify the expression above to get $\mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]^Q_Z > U$. □

A.2 Proof of Theorem 4

Before proceeding to prove the theorem, we establish the following lemma:

Lemma 5. Let $\varphi(v)$ be an unary logic predicate, and P be a partial model. Then $\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]^P_{abs\circ Z} \leq \#_v^{1/2}[\![\varphi]\!]^P_Z$.

Proof. Consider the sets

$$A = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^Q_{Z, v\mapsto x} \geq 1/2\},$$
$$B = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^Q_{Z, v\mapsto x} = 1\},$$
$$C = \{x \in \mathcal{O}_Q \mid [\![\varphi]\!]^Q_{Z, v\mapsto x} = 1/2\},$$

where $A = B \cup C$. Then $\mathcal{S}_P \vDash \sum\{\widehat{x} \mid x \in C\} \geq 0$ and

$$\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]^P_Z = \sum\{\widehat{x} \mid x \in B\} \leq \sum\{\widehat{x} \mid x \in B\} + \sum\{\widehat{x} \mid x \in C\} = \sum\{\widehat{x} \mid x \in A\} = \#_v^{1/2}[\![\varphi]\!]^P_Z.$$

Now we can prove

$$\mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]^Q_Z \geq L \implies \mathcal{S}_P \nvDash \#_v^{1/2}[\![\varphi]\!]^P_{abs\circ Z} < L$$

by contradiction. Let us assume that $\mathcal{S}_P \vDash \#_v^{1/2}[\![\varphi]\!]^P_{abs\circ Z} < L$. By Theorem 3, we also have $\mathcal{S}_Q \vDash \#_v^{1/2}[\![\varphi]\!]^Q_Z < L$. Hence

$$\mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]^Q_Z \leq \#_v^{1/2}[\![\varphi]\!]^Q_Z < L$$

by Lemma 5. We simultaneously have

$$\mathcal{S}_Q \vDash L \leq \#_v^{1}[\![\varphi]\!]^Q_Z < L,$$

which is a contradiction. $\mathcal{S}_Q$ cannot be satisfiable.

The case of

$$\mathcal{S}_Q \vDash \#_v^{1/2}[\![\varphi]\!]^Q_Z \leq U \implies \mathcal{S}_P \nvDash \#_v^{1}[\![\varphi]\!]^P_{abs\circ Z} > U$$

is analogous. After assuming $\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]^P_{abs\circ Z} > U$, we get the contradiction

$$\mathcal{S}_Q \vDash U \geq \#_v^{1/2}[\![\varphi]\!]^Q_Z \geq \#_v^{1}[\![\varphi]\!]^Q_Z > U$$

by applying Theorem 3 and Lemma 5. □

B Mean execution times of model generators

Tables 4–7 show the mean execution times of model generators from **RQ3** and **RQ4**. Only the m executions within time and memory limits (i.e., successfully terminating) are counted out of the 30 runs performed. The mean execution time t of the fastest approach is highlighted in **bold**. The • symbol shows statistically significant differences in mean execution times of the baseline **A/S4J**, **A/MS**, **GS/O** approaches from our proposed **GS/S** according to a two-tailed t-test at significance level $p < 0.05$. The ∘ symbol shows differences that are not statistically significant, and ? denotes cases where there were too few successful runs for significance testing.

In Table 4 (**RQ3.1**), **GS/S** significantly outperforms all baseline approaches in all domains for large model sizes ($n \geq 20$ for **Sat**, $n \geq 80$ for **Sct**, $n \geq 30$ for **Met**).

For model generation problems without type scope constraints in Table 5 (**RQ3.2**), **GS/S** outperformed the baseline approaches in **Sat**. In **Sct**, the difference between **GS/O** and **GS/S** was not statistically significant for large model

TABLE 4
Mean execution times of model generator runs finished within the time limit with scope constraints (+**S**)

		A/S4J		**A/MS**		**GS/O**		**GS/S**	
	n	t/s	m	t/s	m	t/s	m	t/s	m
SAT	10	0.82 •	30	**0.51** •	30	16.48 •	30	3.13	30
	20	49.08 •	30	10.68 ○	30	15.46 ○	30	**7.73**	30
	30	169.49 •	2	70.30 •	30	66.34 •	18	**9.42**	30
	40	–	0	263.68 •	4	48.05 •	18	**6.82**	30
	50	–	0	–	0	42.82 •	14	**15.08**	30
	60	–	0	–	0	83.94 •	17	**13.17**	30
	70	–	0	–	0	137.58 •	10	**30.45**	30
	80	–	0	–	0	218.69 •	13	**30.77**	29
	90	–	0	–	0	269.55 •	3	**34.78**	30
	100	–	0	–	0	–	0	**61.68**	30
SCT	20	0.24 •	30	**0.22** •	30	120.43 •	15	1.59	30
	40	2.07 •	30	**1.66** •	30	–	0	3.03	30
	60	8.29 ○	30	**5.91** •	30	–	0	7.46	30
	80	26.16 •	30	16.64 •	30	–	0	**7.89**	30
	100	43.56 •	30	34.38 •	30	–	0	**16.68**	30
	120	125.35 •	30	63.59 •	30	–	0	**13.36**	30
	140	195.25 •	30	120.54 •	30	–	0	**18.71**	30
	160	299.08 ?	1	193.96 •	30	–	0	**25.99**	30
	180	–	0	277.94 •	17	–	0	**27.61**	30
	200	–	0	–	0	–	0	**48.62**	30
MET	10	0.15 •	30	**0.14** •	30	6.10 ○	30	4.02	30
	20	1.93 •	30	**1.57** •	30	5.97 •	27	4.42	30
	30	10.90 •	30	8.37 •	30	8.74 •	29	**5.92**	29
	40	34.40 •	30	27.90 •	30	20.42 •	29	**10.79**	29
	50	96.23 •	30	76.16 •	30	42.01 •	29	**26.68**	29
	60	239.80 ○	2	148.63 •	30	35.38 •	30	**20.17**	28
	70	–	0	284.31 •	3	71.50 •	29	**46.94**	29
	80	–	0	–	0	131.19 •	30	**91.88**	29
	90	–	0	–	0	209.29 •	21	**155.42**	29
	100	–	0	–	0	230.81 ○	6	**229.92**	17

n = mode size (number of objects)
t/s = execution time (seconds)
m = number of successful runs within 300 s time limit
• = statistically significant ($p < 0.05$) difference from **GS/S**
○ = difference not deemed statistically significant
? = too few successful runs to test significance

TABLE 5
Mean execution times of model generator runs finished within the time limit without scope constraints (−**S**)

		A/S4J		**A/MS**		**GS/O**		**GS/S**	
	n	t/s	m	t/s	m	t/s	m	t/s	m
SAT	10	0.57 •	30	**0.49** •	30	25.43 •	30	3.33	30
	20	26.80 •	30	12.41 •	30	4.26 ○	30	**3.90**	30
	30	198.28 •	12	133.57 •	30	10.16 •	30	**4.48**	30
	40	–	0	224.25 ?	1	19.40 •	30	**5.52**	30
	50	–	0	–	0	46.03 •	30	**7.81**	30
	60	–	0	–	0	77.95 •	30	**11.25**	30
	70	–	0	–	0	156.80 •	30	**13.49**	30
	80	–	0	–	0	240.75 •	23	**24.38**	30
	90	–	0	–	0	–	0	**27.38**	30
	100	–	0	–	0	–	0	**48.10**	30
SCT	50	4.36 •	30	3.00 •	30	**2.81** •	30	3.56	30
	100	34.12 •	30	30.21 •	30	**5.68** •	30	8.27	30
	150	159.60 •	30	128.70 •	30	**11.78** •	30	16.06	30
	200	–	0	–	0	**20.42** •	30	25.43	30
	250	–	0	–	0	**36.36** ○	30	41.03	30
	300	–	0	–	0	**46.33** •	30	60.73	30
	350	–	0	–	0	**69.97** ○	30	76.54	30
	400	–	0	–	0	**94.14** ○	29	98.91	30
	450	–	0	–	0	**119.28** ○	29	127.17	29
	500	–	0	–	0	**153.39** ○	30	167.28	30
MET	200	–	0	–	0	**7.26** •	30	10.70	30
	400	–	0	–	0	**14.74** •	30	20.54	30
	600	–	0	–	0	**24.79** •	30	33.91	30
	800	–	0	–	0	**37.98** •	30	48.81	30
	1000	–	0	–	0	**54.48** •	30	68.97	30
	1200	–	0	–	0	**82.49** •	30	99.04	30
	1400	–	0	–	0	**99.07** •	30	116.92	30
	1600	–	0	–	0	**116.38** •	30	164.20	29
	1800	–	0	–	0	**149.26** •	28	177.83	28
	2000	–	0	–	0	**168.51** •	25	209.96	24

n = mode size (number of objects)
t/s = execution time (seconds)
m = number of successful runs within 300 s time limit
• = statistically significant ($p < 0.05$) difference from **GS/S**
○ = difference not deemed statistically significant

sizes ($n > 350$). In **MET**, the overhead of **GS/S** in problems with no numerical constraints was apparent, as it was outperformed by **GS/O**.

For problems unsatisfiable due to well-formedness constraints in Table 6 (**RQ4**), **GS/S** significantly outperformed **GS/O**, but was significantly outperformed by both **A/S4J** and **A/MS**. This highlights that, while scope analysis may allow state-space exploration based model generators to remain competitive for proving unsatisfiability for smaller problems, SAT-solvers are much better suited for this task.

For problems unsatisfiable due to scope constraints in Table 7, the execution time of **GS/S** remained constant over various target model sizes (except for $n = 12$ and 13 for **SAT** and $n = 8$ for **SCT** due to rounding artifacts). Scope analysis could quickly reject the inconsistent scope constraints without exploring the state space of the model generation task. Thus, **GS/S** could significantly outperform all baseline approaches.

C Diversity measurement

Setup. To evaluate the structural diversity of the generated models, we used a neighbourhood-based [77] internal diversity metric [20] which correlates with mutation score in mutation testing scenarios. In summary, this metric calculates the proportion of different local neighborhoods of nodes included in a graph model.

We used a neighborhood range = 3, which classifies two objects to be identical, if they cannot be distinguished with at most 3 navigations (hops). To measure structural diversity, the values of data objects are not taken into account. We measured the diversity of 10–10 models for all three case studies (**SAT**, **SCT** and **MET**) with 50 objects with all solvers. For domain **SCT+S** the original solver **GS/O** was not able to generate models, so we left it blank.

Analysis of Results: The distribution of internal diversity is illustrated in Fig. 1. For each domain, each solver approach produced similar internal diversity, where the

TABLE 6
Mean execution times of model generator runs finished within the time limit when proving unsatisfiability due to contradictory well-formedness constraints ($+\lightning_{\text{WF}}$)

		A/S4J		A/MS		GS/O		GS/S	
	n	t/s	m	t/s	m	t/s	m	t/s	m
SAT	5	0.06 •	30	**0.06** •	30	2.65 •	30	2.49	30
	6	0.09 •	30	**0.09** •	30	3.00 •	30	2.48	30
	7	0.15 •	30	**0.15** •	30	6.17 •	30	2.49	30
	8	0.24 •	30	**0.23** •	30	19.81 •	30	2.47	30
	9	0.37 •	30	**0.37** •	30	87.83 •	30	2.47	30
	10	0.58 •	30	**0.54** •	30	–	0	39.08	30
	11	0.86 •	30	**0.79** •	30	–	0	147.85	30
	12	1.27	30	**1.12**	30	–	0	–	0
	13	1.76	30	**1.57**	30	–	0	–	0
	14	2.42	30	**2.10**	30	–	0	–	0
	15	3.38	30	**2.83**	30	–	0	–	0
SCT	5	0.02 •	30	**0.02** •	30	2.85 •	30	1.32	30
	6	0.03 •	30	**0.02** •	30	11.01 •	30	1.50	30
	7	0.03 •	30	**0.03** •	30	60.31 •	30	2.55	30
	8	0.04 •	30	**0.04** •	30	–	0	7.03	30
	9	0.05 •	30	**0.05** •	30	–	0	106.88	30
	10	**0.07**	30	0.07	30	–	0	–	0
	11	0.09	30	**0.09**	30	–	0	–	0
	12	**0.11**	30	0.11	30	–	0	–	0
	13	0.15	30	**0.14**	30	–	0	–	0
	14	**0.18**	30	0.18	30	–	0	–	0
	15	**0.22**	30	0.22	30	–	0	–	0
MET	5	0.04	30	**0.03**	30	–	0	–	0
	6	0.06	30	**0.05**	30	–	0	–	0
	7	0.25	30	**0.10**	30	–	0	–	0
	8	0.88	30	**0.41**	30	–	0	–	0
	9	6.97	30	**4.70**	30	–	0	–	0
	10	49.93	30	**39.29**	30	–	0	–	0
	11	–	0	**274.72**	26	–	0	–	0

n = mode size (number of objects)
t/s = execution time (seconds)
m = number of successful runs within 300 s time limit
• = statistically significant ($p < 0.05$) difference from **GS/S**
○ = difference not deemed statistically significant
🟨 = the minimum model size in the SAT domain, even without the added unsatisfiable $+\lightning_{\text{WF}}$ well-formedness constraints, is 10

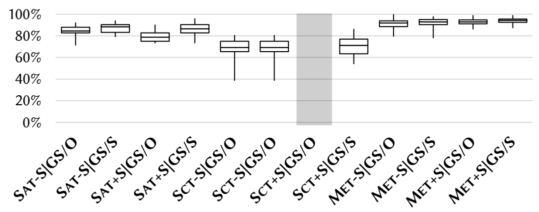


Fig. 1. Internal Diversity distributions

TABLE 7
Mean execution times of model generator runs finished within the time limit when proving unsatisfiability due to contradictory scope constraints ($+\lightning_{\text{S}}$)

		A/S4J		A/MS		GS/O		GS/S	
	n	t/s	m	t/s	m	t/s	m	t/s	m
SAT	5	0.06 •	30	**0.05** •	30	2.55 •	30	2.39	30
	6	**0.09** •	30	0.09 •	30	2.93 •	30	2.36	30
	7	**0.14** •	30	0.14 •	30	4.87 •	30	2.38	30
	8	**0.22** •	30	0.22 •	30	10.27 •	30	2.38	30
	9	0.36 •	30	**0.35** •	30	24.59 •	30	2.37	30
	10	0.55 •	30	**0.54** •	30	55.51 •	30	2.36	30
	11	0.95 •	30	**0.77** •	30	–	0	2.38	30
	12	1.83 •	30	**1.17** •	30	–	0	52.25	30
	13	2.58 •	30	**1.73** •	30	–	0	56.36	30
	14	3.92 •	30	**2.24** •	30	–	0	2.37	30
	15	4.51 •	30	3.11 •	30	–	0	**2.37**	30
	20	19.90 •	30	10.45 •	30	–	0	**2.43**	30
	30	241.28 •	13	77.92 •	30	–	0	**2.42**	30
	40	–	0	–	0	–	0	**2.44**	30
	50	–	0	–	0	–	0	**2.41**	30
	60	–	0	–	0	–	0	**2.41**	30
	70	–	0	–	0	–	0	**2.42**	30
	80	–	0	–	0	–	0	**2.44**	30
	90	–	0	–	0	–	0	**2.42**	30
	100	–	0	–	0	–	0	**2.44**	30
SCT	5	0.02 •	30	**0.02** •	30	1.72 •	30	1.17	30
	6	0.02 •	30	**0.02** •	30	2.99 •	30	1.16	30
	7	0.03 •	30	**0.02** •	30	9.50 •	30	1.16	30
	8	0.04 •	30	**0.03** •	30	2.56 •	30	1.33	30
	9	0.04 •	30	**0.03** •	30	119.90 •	30	1.14	30
	10	0.05 •	30	**0.04** •	30	–	0	1.15	30
	11	0.06 •	30	**0.05** •	30	–	0	1.15	30
	12	0.08 •	30	**0.06** •	30	–	0	1.16	30
	13	0.09 •	30	**0.07** •	30	–	0	1.15	30
	14	0.09 •	30	**0.08** •	30	–	0	1.15	30
	15	0.13 •	30	**0.10** •	30	–	0	1.14	30
	20	0.28 •	30	**0.22** •	30	–	0	1.18	30
	40	2.40 •	30	1.62 •	30	–	0	**1.19**	30
	60	7.65 •	30	5.60 •	30	–	0	**1.18**	30
	80	22.28 •	30	15.52 •	30	–	0	**1.18**	30
	100	47.26 •	30	32.18 •	30	–	0	**1.18**	30
	120	83.39 •	30	57.16 •	30	–	0	**1.18**	30
	140	143.72 •	30	106.49 •	30	–	0	**1.18**	30
	160	250.45 •	30	164.96 •	30	–	0	**1.18**	30
	180	–	0	246.11 •	30	–	0	**1.17**	30
	200	–	0	–	0	–	0	**1.18**	30

n = mode size (number of objects)
t/s = execution time (seconds)
m = number of successful runs within 300 s time limit
• = statistically significant ($p < 0.05$) difference from **GS/S**
○ = difference not deemed statistically significant
🟫 = **GS/S** required state space exploration in the SAT domain for models of size 12 and 13
🟩 = the model generation problem in the SCT domain was satisfiable for size 8

internal diversity of models generated with the **GS/S** approach was slightly better.

Our approach provides similarly high structural diversity as the original solver, with or without scope constraints.

Based on the findings from [20], these results indicate that the models generated with scope constraints (+**S**) can serve as at least as good test cases for systems engineering tools as those generated without the scope constraints (−**S**). Moreover, results on the +**S** models may be easier to interpret, because they use elements that more commonly occur in practice.