ABSTRACT

Automatically synthesizing consistent models is a key prerequisite for many testing scenarios in autonomous driving or software tool validation where model-based systems engineering techniques are frequently used to ensure a designated coverage of critical corner cases. From a practical perspective, an inconsistent model is irrelevant as a test case (e.g., false positive), thus each synthetic model needs to simultaneously satisfy various structural and attribute well-formedness constraints. While different logic solvers or dedicated graph solvers have recently been developed, they fail to handle either structural or attribute constraints in a scalable way.

In the current paper, we combine a structural graph solver that uses partial models with an SMT-solver to automatically derive models which simultaneously fulfill structural and attribute constraints while key theoretical properties of model generation like completeness or diversity are still ensured. This necessitates a sophisticated bidirectional interaction between different solvers which carry out consistency checks, decision, unit propagation, concretization steps. We evaluate the scalability and diversity of our approach in the context of three complex case studies.

CCS CONCEPTS
• Software and its engineering → Domain specific languages;
• Mathematics of computing → Solvers.

KEYWORDS
model generation, partial modeling, SMT-solvers
Recent model generators [56, 63, 64] have successfully improved on scalability by lifting the model synthesis problem on the level of graph models by using meta-heuristic search [63] possibly combined with an SMT-solver [64]. Alternatively, partial model refinement [56] can be used as search strategy while efficient query/constraint evaluation engines [66, 68] validate the constraints during state space exploration. However, there are also important restrictions imposed by these tools such as lack of completeness [63, 64] or lack of attribute handling [56] in constraints.

In this paper, we propose a model generation technique which can automatically derive consistent graph models that satisfy both structural and attribute constraints. For that purpose, the structural constraints are satisfied along partial model refinement (like [56]) while attribute constraints are satisfied by repeatedly calling the Z3 SMT-solver [12] (like [64]). However, as a conceptual extension to preceding work, we define refinement units (in analogy with an abstract DPLL procedure modulo theories (Davis–Putnam–Logemann–Loveland) [39] or SMT-solvers [38]) with consistency checking, decision, unit propagation and concretization steps to enable a bidirectional interaction between a graph solver and an SMT-solver where a decision in one solver can be propagated to the other solver and vice versa. In particular,

- We define 3-valued logic semantics for evaluating structural and attribute constraints over partial models.
- We propose qualitative abstractions to uniformly represent attribute constraints as (structural) relations in a model.
- We define a mapping from attribute constraints to an SMT-problem interpreted by an SMT-solver.
- We propose a generic model generation process with bidirectional interaction between structural and attribute solvers. We present the detailed description of an attribute solver for numeric (e.g. int or double) constraints.
- We evaluate a prototype implementation of the approach on three case studies to assess scalability and diversity properties of model generation.

As a main added value with respect to existing results, our approach provides good scalability for automatically generating consistent models with structural and attribute constraints while still providing completeness and diversity.

## 2 PRELIMINARIES

### 2.1 Running example

We illustrate the challenges of handling both structural and attribute constraints in model generation for a simple domain of family trees with a metamodel shown in Figure 1 and well-formedness (WF) constraints defined by graph patterns (in the VQL language [67, 68]) listed in Figure 2. This domain is intentionally chosen to contain only few domain concepts, while it can demonstrate all key technical challenges of constraint evaluation.

A FamilyTree contains Members with an integer age attribute. Members are related to each other by parents relations. The violating cases of the three WF constraints are defined by VQL graph patterns that all consistent family tree models need to respect:

- twoMembersHaveNoParent: There is at most one member in a family tree without a parent;
- negativeAge: All age attributes of family members are non-negative numbers;
- parentTooYoung: There must be more than 12 years of difference between the age attribute of a parent and a child.

In this paper, we use color coding to separate logic and numeric reasoning. The first constraint is a structural constraint (i.e. only navigation along object references) while the second constraint is a numerical constraint which accesses the age attribute (mAge) of a family member m and checks if the value of age is negative. The third constraint contains both structural and numerical clauses which mutually depend on each other. (1) If a new parents reference is created between two family members then a new numerical constraint needs to be enforced between the respective age attributes (logic→numeric dependency). (2) If the age attribute of two family members is already determined then a new parents reference may (or must not) be added between them (numeric→logic dependency).

Our paper investigates how to generate consistent models in the presence of such mutual dependencies between structural and attribute constraints. The bidirectional interaction is exemplified in the paper for numerical attributes, but the conceptual framework is applicable to attributes of other domains (e.g. strings, bitvectors) assuming the existence of an underlying solver (e.g. SMT-solver) for the background theory of the respective attribute. Our model generation framework semantically relies on model refinement carried out by 3-valued constraint evaluation over partial models.

### 2.2 Domain-specific partial models

**Domain specification.** We formalize the concepts in a target domain (Σ, α) using an algebraic representation with a signature Σ and an arity function α : Σ → ℕ. Such a signature Σ = {T₁, ..., Tₙ, R₁, ..., Rₚ, P₁, ..., Pₚ, A₁, ..., Aₕ, ᵇ, ~} can be easily derived from metamodeling formalisms like EMF [65].

- Unary predicate symbols (T₁, ..., Tₙ) (with α(Tᵢ) = 1) are defined for each EClass and EEnum in the domain (e.g. FamilyTree). Bool denotes the EBoolean type, Int denotes integer numbers types like EInt or EShort, etc.
- Binary predicate symbols (R₁, ..., Rₚ) (with α(Rᵢ) = 2) are defined for each EReference and EAttribute in the metamodel.
For example, parents represent the parent reference between two Members, and age represents the age attribute relation between a Member and an Elnt.

- Structural predicate symbols \( \{ P_1, \ldots, P_p \} \) are n-ary predicates derived from graph queries (with \( \alpha(P_\iota) = n \) equal to the number of formal parameters of a graph query); e.g. parentToYoung becomes a binary predicate symbol.
- Attribute predicate symbols \( \{ A_1, \ldots, A_q \} \) represent n-ary predicates derived from attribute (check) expressions of queries (with \( \alpha(A_\iota) = n \); e.g. check<p>\#12(c, p) is a binary attribute predicate with parameters c and p.
- A special unary symbol \( \& \) denotes the existence of objects.
- A special binary symbol \( \sim \) denotes the equivalence relation between two objects, which can be represented explicitly.

**Partial models.** Partial models are frequently used to explicitly represent uncertainty in models [13, 50], which is particularly relevant for intermediate steps of a model generation process. We use 3-valued partial models where the traditional truth values true (1) and false (0) are extended with a third truth value 1/2 to denote unknown structural parts of the model [19, 47, 57]. Similarly, we extend the domain of traditional numeric values (e.g. 1 or 2.1) with ? to denote an unknown numeric value.

**Definition 2.1.** Given a signature \( (\Sigma, \alpha) \), a numerical partial model is a logic structure \( P = \{ O_P, I_P, V_P \} \) where:

- \( O_P \) is the finite set of objects in the model,
- \( I_P \) gives a 3-valued logic interpretation for each symbol \( s \in \Sigma \) as \( I_P(s): O_P^{\alpha(s)} \rightarrow \{ 0, 1, 1/2 \} \),
- \( V_P \) gives a numeric value interpretation for each object in the model: \( V_P: O_P \rightarrow \mathbb{R} \cup \{ ? \} \).

Note that this definition uniformly handles domain objects (e.g. Member) and data objects (e.g. Int), which is frequently the case in object-oriented languages. Next, we capture some regularity restrictions to exclude irrelevant (irregular) partial models:

**Definition 2.2.** A partial model \( P = \{ O_P, I_P, V_P \} \) is regular, if it satisfies the following conditions:

R1 \( \forall o \in O_P: I_P(o) > 0 \) (non-existing objects are omitted)
R2 \( \forall o \in O_P: I_P(\neg)(o) > 0 \) (\( \neg \) is reflexive)
R3 \( \forall o_1, o_2 \in O_P: I_P(\neg)(o_1, o_2) = I_P(\sim)(o_1, o_2) \) (\( \sim \) is symmetric)
R4 \( \forall o_1, o_2 \in O_P: (o_1 \neq o_2) \Rightarrow I_P(\sim)(o_1, o_2) < 1 \) (if two objects are different, then they cannot be equivalent)
R5 \( \forall o \in O_P: (\{ I_P(\text{Int})(o) = 0 \} \lor (I_P(\text{Real})(o) = 0)) \Rightarrow (\{ V_P(o) = ? \}) \) (domain objects do not have a value)
R6 \( \forall o \in O_P: (\{ V_P(o) = ? \}) \Rightarrow (\{ I_P(\text{Int})(o) = 1 \} \lor (I_P(\text{Real})(o) = 1)) \) (objects with values are numbers)
R7 \( \forall o \in O_P: (\{ I_P(\text{Int})(o) = 1 \}) \Rightarrow (\{ V_P(o) = ? \} \lor (V_P(o) \in \mathbb{N})) \) (only natural numbers are bound to Int objects)

**Example 2.3.** Figure 6 illustrates partial models. In State \( 1 \), we have three concrete objects (where \( \& \) and \( \sim \) are 1): FamilyTree \( f \) and a Member \( m \), and an unbound Int data object \( a \) (with ? value). The partial model also contains an abstract “new objects” node that represents multiple potential new nodes (using 1/2 values for \( \& \) and for \( \sim \), denoted by dashed border), and a “new integers” node representing the potential new integers. In Figure 6, predicates with value 1 are denoted by solid lines (as for the member edge between \( f \) and \( m \) in State \( 1 \)) and predicates with value 1/2 are denoted by dashed lines (like the potential parents edge in State \( 1 \)).

### 2.3 Refinement and concretization

During model generation, the level of uncertainty in partial models will be gradually reduced by refinements. In a refinement step, uncertain 1/2 values can be refined to either 1 or 0, or unbound values are refined to concrete numerical values. This is captured by an information ordering relation \( X \subseteq Y := (X = 1/2) \lor (X = Y) \) where an \( X = 1/2 \) is either refined to another value \( Y \), or \( X = Y \) remains equal. An information ordering can be defined between numerical values \( x \) and \( y \) similarly \( x \subseteq Y := (x = ?) \lor (x = y) \).

A refinement from partial model \( P \) to partial models \( Q \) is a mapping that respects both information ordering relations \( \subseteq \).

**Definition 2.4.** A refinement from regular partial model \( P \) to regular partial model \( Q \) (denoted as \( P \subseteq Q \)) is defined by a refinement function between the objects of the partial model \( \text{ref}: O_P \rightarrow 2^{O_Q} \) which respect the information ordering:

- For each n-ary symbol \( s \in \Sigma \), each object \( o_1, \ldots, o_n \in O_P \), and for each refinement \( q_1 \in \text{ref}(p_1), \ldots, q_n \in \text{ref}(p_n) \):
  \( I_Q(s)(q_1, \ldots, q_n) \).
- For each object \( p \in O_P \) and its refinement \( q \in \text{ref}(p) \):
  \( V_Q(p) \subseteq \text{ref}(q) \).
- All objects in \( Q \) are refined from an object in \( P \), and existing objects \( p \in O_P \) must have a non-empty refinement.

Model generation is carried out along refinements by eventually resolving all uncertainties to obtain a concrete model.

**Definition 2.5.** A regular (see Definition 2.2) partial model \( P \) is concrete, if (a) \( I_P \) does not contain 1/2 values, and (b) \( V_P \) does not contain ? values for integer and real data objects (for object \( o \) where \( I_P(\text{Int})(o) = 1 \) or \( I_P(\text{Real})(o) = 1 \)).

**Example 2.6.** Figure 6 illustrates several refinement steps. Between State \( 0 \) and State \( 1 \), new object is split into two objects by refining \( \& \) to 0 between new object and \( m \), creating a single concrete object \( m \) by refining \( \sim \) on \( m \) to 1. Simultaneously, type Member is refined to 1, FamilyTree refined to 0, and reference predicate members from \( f \) to \( m \) is refined to 1. Eventually, the value of data object \( a \) is refined from ? to 2 in State \( 4.2 \).

### 2.4 Constraints over partial models

**Syntax.** Both structural (logical) and numerical constraints can be evaluated on partial models. For each graph pattern we derive a logic predicate (LP) defined as \( P(a_1, \ldots, a_n) \Leftrightarrow \phi \), where \( \phi \) is a logic expression (LE) constructed inductively from the pattern body as follows (assuming the standard precedence for operators).

- if \( s \in \Sigma \) is an n-ary predicate symbol (i.e. \( T, R, P, A, \& \) or \( \sim \) then \( s(a_1, \ldots, a_n) \) is a logic expression;
- if \( \phi_1 \) and \( \phi_2 \) are logic expressions, then \( \phi_1 \lor \phi_2, \phi_1 \land \phi_2, \text{and} \quad \neg \phi_1 \text{are logic expressions;}
- if \( \phi \) is a logic expression, and \( v \) is a variable, then \( \exists v: \phi \) and \( \forall v: \phi \) are logic expressions.
For each attribute constraint, we derive attribute predicates (as helpers) by reification to enable seamless interaction between structural and attribute solvers along a compatibility (if and only if) operator $\Leftarrow$ (see Figure 3). In case of numbers, such an attribute predicate is tied to a numerical predicate defined as $\text{A}(v_1, \ldots, v_n) \Leftarrow \psi$ where $\psi$ is constructed from numeric expressions. The expressiveness of those expressions are limited by the background theories of the underlying backend SMT solver. Here we define a core language of basic arithmetical expressions, which is supported by a wide range of numerical solvers:

- each variable $v$, constant symbol and literal (concrete number) $c$ is a numerical expression,
- if $\psi_1$ and $\psi_2$ are numerical expressions, then $\psi_1 + \psi_2, \psi_1 - \psi_2, \psi_1 \times \psi_2$ and $\psi_1 \div \psi_2$ are numerical expressions,
- if $\psi_1$ and $\psi_2$ are numerical expressions then $\psi_1 < \psi_2, \psi_1 > \psi_2, \psi_1 \geq \psi_2, \psi_1 \leq \psi_2, \psi_1 = \psi_2, \psi_1 \neq \psi_2$ are numerical predicates.

Example 2.7. Graph pattern $\text{parentTooYoung}(\text{child}, \text{parent})$ from Figure 2 is formalized as the following logic predicate:

$$\text{parentTooYoung}(\text{child}, \text{parent}) \Leftarrow \text{parents}(\text{child}, \text{parent}) \land \text{age}(\text{child}, c) \land \text{age}(\text{parent}, p) \land \text{check}_{P \leq c + 12}(c, p)$$

where $\text{check}_{P \leq c + 12}(c, p) \Leftarrow P \leq c + 12$ is a numerical predicate.

Later such predicates will help communicate between different solvers, e.g. if $\text{check}_{P \leq c + 12}(c_1, p_1)$ is found to be true by the graph solver for some members $c_1$ and $p_1$ then the numerical predicate $P_1 \leq c_1 + 12$ needs to be enforced by a numerical solver for the respective data objects and vice versa.

**Semantics.** A logic predicate $P(v_1, \ldots, v_n) \Leftarrow \psi$ can be evaluated on a partial model $P$ along a variable binding $Z : \{v_1, \ldots, v_n\} \rightarrow OP$ (denoted as $[\psi]_Z^P$), which can result in three truth values: 1, 0 or $1/2$.

The inductive rules of evaluating the semantics of a logic expression is illustrated in Figure 3. Note that $\min$ and $\max$ takes the numerical minimum and maximum values of 1, 0 or $1/2$.

A numerical predicate $A(v_1, \ldots, v_n) \Leftarrow \psi$ can also be evaluated on a partial model $P$ along variable binding $Z : \{v_1, \ldots, v_n\} \rightarrow OP$ (denoted as $[\psi]_Z^P$) with a result of 1, 0 or $1/2$. The inductive rules capturing the semantics of logic expressions are illustrated in Figure 3.

Note that $\langle \text{cmp} \rangle$ $y$ means the truth value of numerical comparison $\langle \text{cmp} \rangle$ (e.g. $3 < 5$ is 1), while $\langle \text{op} \rangle$ $y$ means the numerical value of the result of an operation $\langle \text{op} \rangle$ (e.g. $3 + 5$ is 8).

**Constraint approximation.** When a predicate is evaluated on a partial model, then the 3-valued semantics of constraint evaluation guarantees that certain (over- and under-approximation) properties hold for all potential refinements or concretizations of the partial model. For all logic and numeric predicates $\varphi$ and $\psi$, if $P \subseteq Q$ then $[\varphi]_P^P \subseteq [\varphi]_Q^Q$ and $[\psi]_P^P \subseteq [\psi]_Q^Q$, consequently:

- **Logic under-approximation**: If $[\varphi]_P^P = 1$ in a partial model $P$ then $[\varphi]_Q^Q = 1$ in any partial model $Q$ where $P \subseteq Q$.
- **Numeric under-approximation**: If $[\varphi]_P^P = 1$ in a partial model $P$ then $[\varphi]_Q^Q = 1$ in any partial model $Q$ where $P \subseteq Q$.
- **Logic over-approximation**: If $[\varphi]_Q^Q = 0$ in a partial model $Q$ then $[\varphi]_P^P \geq 1/2$ in a partial model $P$ where $P \subseteq Q$.
- **Numeric under-approximation**: If $[\varphi]_Q^Q = 0$ in a partial model $Q$ then $[\psi]_P^P \leq 1/2$ in a partial model $P$ where $P \subseteq Q$.

Using these properties, model generation becomes a monotonous derivation sequence of partial models which starts from the most abstract partial model where all predicate constraints are evaluated to $1/2$. The partial model is gradually refined, thus more and more predicate values are evaluated to either 1 or 0. The under-approximation lemmas ensure that when an error predicate is evaluated to 1 it will remain 1, thus exploration branch can be terminated without loss of completeness [69]. The over-approximation lemmas assure that if a partial model can be refined to a concrete model where error predicate is 0, then it will not be dropped.

### 3 MODEL GENERATION WITH REFINEMENT

#### 3.1 Functional overview

Our model generation approach takes the following inputs:

1. The signature of a domain $(\Sigma, \alpha)$ with structural logic symbols $P_1, \ldots, P_p$ and numerical attribute symbols $A_1, \ldots, A_n$.
2. A logic theory consisting of the negation of the error predicates and the compatibility of the predicate symbols with their definition (i.e. the axioms): $\mathcal{T} = \{ \neg E_1, \ldots, \neg E_r, (P_1 \Leftarrow \varphi_1^P), \ldots, (P_p \Leftarrow \varphi_p^P), (A_1 \Leftarrow \varphi_1^A), \ldots, (A_n \Leftarrow \varphi_n^A)\}$
3. Some search parameters (e.g., the required size, or the required number of models).
where each $M$ (based on the structural part of the error predicates), which need to
be respected by the numerical refinement unit. Symmetrically, the structural refinement unit refine truth values on attribute predicates (based on the structural part of the error predicates), which need to be respected by the numerical refinement unit in turn.

3.2 State space exploration by refinements

Our model generation framework derives models by exploring the search space of partial models along refinements carried out by refinement units. As such, the size of the partial models is continuously growing up to a designated size, while the exploration process tries to intelligently minimize the search space. The detailed steps of this exploration process are illustrated in Figure 5.

0. Initialization: First, we initialize our search space with an initial partial model. This is derived either from an existing initial model provided by an engineer (thus each solution will contain this seed model as a submodel), or it can be the most general partial model $P_0 = \langle O_{P_0}, \ I_{P_0}, \ V_{P_0} \rangle$ where $O_{P_0} = \{ \text{new} \}$ has a single element, $V_{P_0}$ is $1/2$ for every symbol, and $V_{P_0}(\text{new}) = 2$.

1. Decision: Next, we select an unexplored decision candidate proposed by a refinement unit, and execute it to refine the partial model by adding new nodes and edges, or by populating a data object with a concrete value. In our setup, this decision step is executed mainly by the structural refinement unit which has more impact on model generation. If no decision candidates are left unexplored, the search concludes with an UNSAT result.

2. Unit propagation: After a decision, the framework executes unit propagation in all refinement units until a fixpoint is reached in order to propagate all consequences of the decision.

3. State coding: The search can reach isomorphic partial models along multiple trajectories. To prevent the repeated exploration of the same state, a state code is calculated and stored for a new partial model by using shape-based graph isomorphism checking [42, 43]. If exploration detects that a partial model has already been explored, it drops the partial model and continues search from another state. Otherwise the framework calculates the state code of the newly explored partial model and continues with its evaluation.

4. Consistency check: Next, each refinement unit checks whether the partial model contains any inconsistencies that cannot be repaired. Structural refinement unit evaluates the (logic) under-approximation of the error predicates (see Section 2.4), which can detect irreparable structural errors. The numerical refinement unit carries out a satisfiability check of the numerical constraint determined by a call to the numerical solver.

5. Concretization: Then the framework tries to concretize the partial model to a fully-defined solution candidate by resolving all uncertainties, and checks its compliance with the target theory and model size. If no violations are found and the model is of given size, then the instance model is stored as a solution. Note that this directly ensures the correctness of all solutions. If this concretization fails, it indicates that something is missing from the model, so the refinement process continues.

6. Approximate distance & Add to state space: When a partial model is refined, our framework estimates its distance from a solution [33]. This heuristic is based on the number of missing objects and the number of violations in its concretization. Then, the new partial model is added to the search space of unexplored decisions where the exploration continues at 1. Decision.
Further heuristics: For selecting the next unexplored decision to refine, we use a combined exploration strategy with best-first search heuristic, backtracking, backjumping and random restarts with an advanced design space exploration framework [21, 56].

Example 3.1. Figure 6 illustrates a model generation run to derive a family tree. Search is initialized with a FamilyTree f1 as root and two (abstract) objects to represent new objects and new integers.

State 1 highlights the execution of a decision refinement that splits the new object and the new integer, creating a new Member m1 with its undefined age attribute.

In State 2.1, a loop parent edge is added as a decision. When investigating error predicate parentTooYoung(child, parent), the search reveals that all conditions of the error predicate are surely satisfied on objects m1 and a1 except for attribute predicate checkP≤+12. Therefore, the structural refinement unit can refine the partial model by setting $f_{S2.1}(\text{check}_{P \leq +12})(a1, a1)$ to 0 without excluding any valid refinements, which implies that $0 \leq c + 12$. The numerical refinement unit (with the help of an underlying SMT solver) can detect that no value $V_{S2.1}(a1)$ can be bound to object a1 such that $V_{S2.1}(a1) \leq V_{S2.1}(a1) + 12$ is false, therefore the model cannot be finished to a consistent model thus it can be safely dropped.

In State 2.2, a new Member m2 is added to the FamilyTree, the framework attempts to concretize the model by resolving all uncertainty in State 3.1. First, the structural refinement unit concretizes in the structural part of the model, all 1/2 values are set to 0 (e.g. all the potential parent edges disappear). Then, sample valid values are generated for the attributes by the numerical refinement unit. When the concretization is checked, error pattern twoMembersHaveNoParent(m1, m2) indicates that there are some missing parent edges, so the framework drops the concretization but continues to explore State 2.2.

Eventually, after adding a parent edge in State 3.2, the framework is able to concretize a model in State 4.2 that satisfies the target theory, thus concluding the search with a consistent model.

3.3 Structural refinements by a graph solver

Our structural refinement unit uses a graph solver [56].

The structural consistency of a partial model can be verified by checking the compatibility of all predicates $P$ as $[P \iff \phi_P]_P$. If a predicate is incompatible with its definition, or an error predicate is satisfied, the partial model is inconsistent (see Section 2.4).

Our framework operates on a graph representation of partial models (without a mapping to a logic solver), thus structural predicates are evaluated directly on this graph representation. The query rewriting technique [57] enables to efficiently evaluate the 3-valued semantics of logic predicates $\{0, 1, \#\}$ by a high-performance incremental model query engine [67, 68], which caches and maintains the truth values of logic predicates during exploration.

Structural refinements are implemented by graph transformations [56, 69]. Decisions are simple transformation rules that rewrite a single $1/2$ value to a 1 in the partial model, or an equivalence predicate $\sim$ to 0 to split an object to two (like m1 is separated from new object in State 1). On the other hand, concretization rewrites all 1/2 values to 0, and self-equivalences to 1.

The compatibility of predicate symbols is checked by structural unit propagation rules, which are derived from error predicates to refine a partial model when needed to avoid a match of an error predicate. We rely on two kinds of unit propagation rules:

- We derive unit propagation rules from the structural constraints imposed by the metamodel to enforce type hierarchy, multiplicities, inverse references, and containment hierarchy [56]. For example, when a new Member is created, a new Int is also created with an age predicate between them.
- From each error predicate $E(v_1, \ldots, v_n)$, unit propagation rules are derived to check when a 1 (or 0) value would satisfy the error predicate $E(v_1, \ldots, v_n)$. In such cases, the value is refined to the opposite 0 (or 1). Such unit propagation rules may add numerical implications of error predicates.

3.4 Numerical refinements by SMT-solvers

The numerical refinement unit is responsible for maintaining the compatibility of numerical constraints and attribute predicates, checking consistency of numerical constraints, and deriving concrete numeric values in the model.

Numerical refinement is based on a purely numerical problem created from a partial model. Let $P$ be a partial model with attribute predicates $A_1, \ldots, A_n$ defined by $\psi^{A_1}, \ldots, \psi^{A_n}$. Let $O^P_{Num}$ denote the set of data objects where $[\text{Int}(o) \lor \text{Real}(o)]_{P_{\text{Num}}}^P \geq 1/2$. For each data object $o \in O^P_{Num}$, we create a numeric variable $V(o)$ denoting its potential value. If $[\text{Int}(o)]_{P_{\text{Num}}}^P \geq 1/2$, then the type of this variable is integer, while if $[\text{Real}(o)]_{P_{\text{Num}}}^P \geq 1/2$ then it is real.

The numerical problem derived from the partial model is defined over these variables. First, if the value of a data object $o \in O^P_{Num}$ is already known in the partial model (i.e. $V^P(o) \neq ?$), then we assert its value as a numerical equation: $V(o) = V(o)$. Next, for each attribute $A_i$, we assert its definition $\psi^{A_i}$ for all data objects:

- If $[\{A_i(v_1, \ldots, v_n)]_{P_{\text{Num}}}^P = 1$, we assert $\psi^{A_i}(V(o_1), \ldots, V(o_n))$
- If $[\{A_i(v_1, \ldots, v_n)]_{P_{\text{Num}}}^P = 0$, we assert $\neg \psi^{A_i}(V(o_1), \ldots, V(o_n))$
An unsatisfiable numerical problem

The entire numerical problem \( \Psi \) as constructed as the conjunction of the respective numerical clauses, which can be solved by an SMT-solver like Z3 [12]. The SMT-solver calculates the satisfiability of the numerical problem as a consistency check for partial models. An unsatisfiable numerical problem \( \Psi \) would imply that the partial model cannot be completed with consistent numerical values, thus it can be dropped.

If the problem is satisfiable, then each variable can be bound to a concrete number (value), and the value assigned to \( V(o) \) is recorded in the partial model as \( V_P(o) \). This step is used to complete partial models by bounding all unbounded data objects during concretization.

Additionally, fixing a potential value for a single data object can be used as a decision. However, in a typical model generation setting, the number of potential values that can be assigned to an object is huge, thus this feature was not used.

The numerical consequences of the constructed \( \Psi \) can be used to refine a partial model during unit propagation. In our framework, three kinds of unit propagation operations are supported:

- When the values of objects \( o_1, \ldots, o_n \) are all known in a partial model, then the truth value of \( \psi^{A_i}(o_1, \ldots, o_n) \big|_{o_1, \ldots, o_n} \) can be calculated in the model without calling an SMT solver.
- If an attribute predicate \( A_i \) has an unknown value \( \big|_{o_1, \ldots, o_n} \) \( \Psi_P \) and \( \psi^{A_i}(o_1, \ldots, o_n) \) is proved to be inconsistent by the SMT-solver, then \( \varphi Q(A_i) \) must be set to \( 0 \) in the refined partial model. Similarly, if \( \Psi_P \land \neg \psi^{A_i}(o_1, \ldots, o_n) \) is inconsistent, the attribute can be refined to 1. In our case studies, this step was impractical thus this feature was not used.
- An unique solution for \( V(o) \) can be used to set \( V_P(o) \).

Example 3.2. We form a numerical problem based on the partial model of State 4.1 in Figure 6. Since \( \text{check}_{P \leq 12} \) is a condition in the error pattern \( \text{parentTooYoung}(\text{child, parent}) \), \( I_{E2.1}(\text{check}_{P \leq 12}) \) must be set to \( 0 \) in the model and the logical operator of \( \text{check}_{P \leq 12} \) should be negated. Since \( \text{Member} \ m_1 \) is a parent of \( \text{Member} \ m_2 \), we have that \( P \rightarrow A_1 \) and \( C \rightarrow A_2 \). It follows that \( a_1 \not\geq a_2 + 12 \). Similarly, since \( \text{Member} \ m_3 \) is a parent of \( \text{Member} \ m_1 \), we have that \( a_3 \not\geq a_1 + 12 \). The numerical problem to solve here is \((a_1 \not\geq a_2 + 12) \land (a_2 \not\geq a_1 + 12) \). Feeding it into an SMT-solver, we will be informed that this problem is unsatisfiable.

3.5 Soundness and Completeness

With the combination of the structural and numerical refinement units, our proposed approach generates models with numerical attributes using partial model refinement. Our approach is sound: it generates consistent solutions only. This is guaranteed by the direct evaluation of the error predicates and compatibility predicates on the final stage of model refinement. Our approach is structurally complete: for a given scope (size), it is able to generate all models with different graph structures, which is ensured by the approximation lemmas in Section 2.4 and in [56, 69]. We intentionally avoid fulfilling numerical completeness, since even simple models could have potentially infinite number of attribute bindings.

4 EVALUATION

We conducted various measurements to address the following research questions:

RQ1: What are the costs and benefits of calling a SMT-solver continuously during exploration or calling it as postprocessing?
RQ2: How do the different exploration steps contribute to the execution time for generating the first model and incrementally generating subsequent models?
RQ3: What is the scalability of model generation when deriving large models with structural and attribute constraints?

RQ4: How structurally diverse are the synthesized models compared to model generation without attribute constraints?

4.1 Target Domains

We perform model generation campaigns in three complex case studies. The target domain artifacts as well as output models and measurement results are available on GitHub (https://github.com/viatra/VIATRA-Generator) and as a virtual machine (https://doi.org/10.5281/zenodo.3950552).

**Fam**: The *FamilyTree* domain is presented in Section 2 as our running example. We use the metamodel shown in Figure 1 which captures parenthood relations and the age of family tree members (with 2 classes, 3 references and 1 numeric attribute). Furthermore, 3 constraints are defined as graph predicates that place structural and numerical restrictions on family tree members. The initial model used for model generation contains a single *FamilyTree* node. While this domain looks simple, there is a subtle mutual dependency between structural and attribute constraints, which provides extra challenges for the interaction of different solvers.

**Sat**: The *Satellite* domain (introduced in [22]) represents interferometry mission architectures used for space mission planning at NASA. Such an architecture consists of collaborating satellites and radio communication between them, which are captured by a metamodel with 15 classes, 5 references and 2 numerical attributes. Additionally, 18 constraints are defined as graph predicates to capture restrictions on collaborating satellites. The initial model contains a single root node as the starting point for model generation.

**Tax**: The *Taxation* domain (used in [63, 64]) represents the personal income tax management application used by the Government of Luxembourg. We reused the original metamodel which contains 54 classes (including 15 Enum classes), 52 relations and 92 attributes, 44 of which are numerical. Additionally, we replicated the OCL constraints used in [64] as graph predicates.

In order to independently replicate the case study of [64] in a pure EMF context with strict containment hierarchy (instead of UML), we include a Resource class in the metamodel that contains instances of the Household class, which was the root class of the original Taxation metamodel. This allows the instantiation of multiple Household instances within the same model generation task. To enforce the same number of objects, we include an initial model containing a predefined number of Household instances and we prevent the generation of further instances of that class as in [64].

**General setup**: To account for warm-up effects and memory handling of the Java 8 VM, an initial model generation task is performed before the actual measurements and the garbage collector is called explicitly between runs. We performed the measurements on an enterprise server.

4.2 RQ1: Integration with an SMT-solver

**Measurement setup**: We compare three model generation approaches that call an SMT-solver in fundamentally different ways:

- **postSMT** (used as a baseline) does not make any SMT solver calls during model generation only as a postprocessing step.
- **contSMT** calls the SMT solver at every model generation step to repeatedly evaluate numerical constraints.
- **qualSMT** qualitatively approximates a numerical constraint with a manually added structural constraint that enforces an acyclic graph structure for families. Then model generation first addresses the structural constraints while the qualitative abstraction is resolved to concrete numerical attribute values in a postprocessing step.

Note that the SMT-solver is not used to derive models satisfying the structural constraints, only the numerical constraints as poor scalability was reported in [4, 54] for structural constraints. For **RQ1**, we perform measurements exclusively in the *Fam* domain, which contains a complex dependency between structural and numerical constraints, thus it is expected to serve as a stress test for SMT-solver calls. We aim to generate models of different size: from 5 to 10 objects with an increment of 1, and from 20 to 100 objects with an increment of 20. The range for numeric values were not bounded a priori. Ten runs are executed for each approach and model size with a timeout of 5 minutes, and the median of runtimes is taken.

**Analysis of results**: Figure 7 compares the execution times for the three approaches. Unsurprisingly, *postSMT* could only generate models of size 5 and 6 (with a failure rate of over 90% for larger models). These figures imply if there is mutual dependency between both numerical and structural constraints in a domain, then the handling of numerical constraints cannot be postponed to a postprocessing phase while focusing exclusively on structural constraints first during model generation as the SMT-solver cannot correct the incompatibilities introduced by an inconsistent structure. Additionally, we notice that *qualSMT* is faster than *contSMT* by a factor of 3 for larger models. This is partly attributed to the use of VIATRA as a back-end solver for the *qualSMT* approach, which has been shown in [4, 56] to perform better than Z3 (used in *postSMT*) for generating model structures. Moreover, future model generation strategies may skip calls to the SMT-solver in certain steps. While the extra structural constraint that enforces an acyclic graph structure for families was added by human intuition, providing such qualitative abstractions of numerical constraints in an automated way is also a promising direction of future research.

```
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{calls.png}
\caption{Calls to SMT-solvers wrt. size of generated models}
\end{figure}
```

---

1. 12 x 2.2 GHz CPU, 64 GiB RAM, CentOS 6, Java 1.8, 12 GiB Heap

**RQ1**: Mutual dependencies between structural and numerical constraints necessitate repeated calls to an SMT-solver during model generation, which cannot be postponed to postprocessing step. Qualitative structural abstractions of numerical constraints may accelerate model generation by introducing an approximate causality.
4.3 RQ2: Cost of exploration phases

Measurement setup: We perform measurements in all three domains to compare runtimes and their distribution between the different phases of model generation. For each domain, we generate models with an increasing minimum model size of 20, 40, 60, 80 and 100. Again, the range for numeric values were not bounded. We exclude larger model sizes to ensure high success rates and to enable cross-domain comparison of execution phases. For the Tax domain, the initial model contains one instance of the Household class for every 20 generated nodes (which is the typical size of a household in the models generated in [64]) to balance the difficulty of model generation regardless of the target model size.

We execute 10 runs per target model size and take the median runtime values. For each model generator run, we aim to produce the first 10 models within a timeout of 1 hour.

Analysis of results: The decompositions of runtime measurements for all three domains are shown in Figure 8a - Figure 8e. Each phase of model generation is represented by a different color. The initialization phase (0.9 seconds for Fam, 3.5 seconds for Sat, 150 seconds for Tax) is a one-time penalty which is proportional to the complexity of the metamodel and the WF constraints.

In the Fam domain, the runtime is dominated by SMT solver calls. This is attributed to the fact that this domain needs to enforce a global structural constraint (acyclicity as discussed in Section 4.2) by solving numerical constraints while the numerical constraints of other domains are dominantly local (e.g. to fill attribute values). However, extra cost of generating subsequent models is low. In the Sat case study, generating the first model takes less than 30 seconds (dominated by the time required for state encoding), but the cost of incrementally generating the next model is relatively larger. For the target model sizes, execution times in the Tax case study are still dominated by the initialization phase due to the large metamodel and numerous constraints of the domain, while the actual model generation and constraint solving phases were rapid.

4.4 RQ3: Scalability of model generation

Measurement setup: We perform measurements in all three domains with increasing model sizes starting from 100 objects with a step size of 50/100 objects and timeout of 1 hour. A single model is generated in each run. A campaign of 10 runs is executed for each measurement point and the median of successful execution times is taken (i.e. that provide a finite model as result within the given time). We terminate the scalability measurement for a case if any of the 10 runs at a particular size fails to output a finite model. For the Tax domain, we provide Household instances as part of the initial model following the 1-to-20 ratio discussed in Section 4.3.

Analysis of results: Measurement results for RQ3 are shown in Figure 8d-Figure 8f. Interestingly, the proposed approach scaled best for the largest metamodel of the Tax case deriving models with 1,100 objects within an hour. Furthermore, we were able to generate models with 1,200 objects within the same time limit with a success rate of 80%. Model generation with 100% success rate scaled up to 300 objects for the Fam and Sat domains. However, root cause of scalability limits was very different (the SMT-solver in Fam and graph solver in Sat). These results also a posteriori validate our choice of including Fam as a case study, which turned out to be the most complex one for assessing the use of SMT-solvers.

RQ3: Our approach was able to generate consistent models with 300 objects for all three case studies within an hour. For the Tax case, scalability is comparable to figures reported in [64] with well over 1000 objects.

4.5 RQ4: Diversity

Measurement setup: To evaluate the structural diversity of the generated models, we used a neighbourhood-based [44] internal diversity metric [55, 58], which correlates with mutation score in mutation testing scenarios. This metric calculates the proportion of different local neighborhoods of nodes included in a graph model. We checked the structural diversity of models only; the measurement and generation of diverse attribute values with the underlying solver requires further research and beyond the scope of the paper.

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 with 100 objects (Fam+N, Sat+N, Tax+N). As a comparison, we generated 10-10 models without bounding the attributes values or respecting the attribute constraints (Fam-N, Sat-N, Tax-N) by the graph solver [56].

Analysis of Results: The distribution of internal diversity is illustrated in Figure 9. Note that Fam+N,Fam-N, Sat+N and Sat-N showed similarly high internal diversity 80%, while diversity values were somewhat lower for Tax+N and Tax-N.

RQ4: Our approach provides similar structural diversity when generating consistent models with structural and numerical constraints compared to the diversity provided by a graph solver [56].

4.6 Threats to validity

Construct validity. We replicated the Tax case study [64] in a new technological context, which involved (1) to create an Ecore metamodel from an equivalent UML diagram and (2) to manually transform the OCL constraints into equivalent VQL graph patterns. The Ecore metamodel was kindly provided to us by the authors of [64], while we validated each replicated OCL constraint by performing manual equivalence checks. We used similar number of Household objects as in [64] and investigated the output models by graph visualization tools to ensure that similar model generation outputs are obtained, but we refrain from direct numerical comparison of execution times due to those technological differences.

Internal Validity. To strengthen internal validity, our experiments include a warm-up run executed prior to the actual measurements to decrease the fluctuation of runtime results caused by the
5 RELATED WORK

We provide an overview of graph generation approaches that derive consistent graphs. We also discuss some key numerical abstractions and decision procedures.

Logic solver approaches. These approaches translate graphs and WF constraints into a logic formula and use underlying solvers to generate graphs that satisfy them. Back-end technologies used for this purpose include SMT solver such as Z3 [26, 52, 70], SAT-based model finders (like Alloy [25]) [3, 5, 9, 23, 30, 35, 54, 59, 60, 62], CSP-solvers [8, 10, 11, 18], theorem provers [4], first-order logic [6], constructive query containment [41], higher-order logic [20] and an incremental query engine [56].

For most of these approaches, scalability is limited to small models/counter-examples. These approaches are either a priori bounded (where the search space needs to be restricted explicitly) or they have decidability issues. Furthermore, handling of numeric constraints is not available for some of these approaches, particularly ones based on SAT-solvers and first-order logic formulations.

Uncertain models. Partial models are similar to uncertain models, which offer a rich specification language [13, 48] amenable to analysis. They provide a more intuitive, 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 [50], or refined by graph transformation rules [49].

Iterative approaches. Iterative approaches generate models by multiple solver calls. An iterative approach is proposed specifically for allocation problems in [29] based on Formula. In [59] models are generated in by calling Alloy in multiple steps, where each step extends the instance model by a few elements. Finally, an
iterative, counter-example guided synthesis is proposed for higher-order logic formulæ in [36]. For these approaches, when scalability evaluation is included, it is limited to 50 nodes.

**Symbolic model generation techniques.** Certain techniques use abstract (or symbolic) graphs for analysis purposes. A tableau-based reasoning method is proposed for graph properties [1, 40, 51], which automatically refines solutions based on WF constraints, and handles the state space in the form of a resolution tree as opposed to a partial model. When scalability evaluation is included, these techniques demonstrated to derive only small graphs (< 10 objects).

Different approaches use abstract interpretation [44], or predicate abstraction [14, 19, 45] 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.

**Hybrid approaches.** These approaches divide the model generation task into multiple sub-tasks and use a different underlying technique to resolve each one. The PLEDGE model generation tool [64] provides such a scalable implementation by combining meta-heuristic search for model structure generation with an SMT-solver based approach for attribute handling. The Evacon tool [24] implements a search-based evolutionary testing approach followed by symbolic execution to generate tests for object-oriented programs. Autograph [52] sequentially combines a tableau-based approach for model structure generation with an SMT-solver based approach for attribute handling. Such approaches combine multiple techniques in a sequential manner, which is a conceptual restriction for mutually dependent structural and numerical constraints. Moreover, none of these techniques assure completeness of model generation. Another category of hybrid approaches involves assessing multiple components of the model generation task in parallel. This requires the implementation of a certain decision procedure such as DPLL(T) [15, 39] to iterate between underlying techniques, or combine them by sharing variables in their proofs [38]. Such decision procedures are presented alongside their associated properties (e.g. soundness and completeness) at an abstract level in [7, 39], which allows for formal reasoning about their implementations. However, those approaches handle graph-based models inefficiently [59, 69], thus the scalability of those techniques is limited.

**Numerical abstractions.** Handling numeric (integer or real) variables and constraints in model generation scenarios requires their abstract interpretation through numerical abstract domains [37, 61]. Numerical abstract domains may be used to summarize object attributes in value analysis of heap programs [14, 31, 34]. Summarized dimensions [19] were introduced to succinctly represent attributes of a potentially unbounded set of objects via multi-objects. This approach enables attribute handling in three-valued partial models, and allows checking for refinements by abstract subsumption [2]. But these approaches do not generate graph models. The uniqueness of our approach lies in combining numerical abstractions with partial models to guarantee soundness and completeness, while generating models with favorable scalability.

**6 CONCLUSIONS**

In this paper, we proposed an automated model generation approach to derive consistent models that satisfy structural and numerical constraints, which necessitates a bidirectional interaction between a graph solver and an SMT-solver. As a conceptual novelty, we proposed so-called refinement units that carry out consistency checking, decision, unit propagation and concretization steps in conceptual analogy with background theories used in SMT-solvers as part of an abstract DPLL procedure [39]. Therefore, refinement units can seamlessly incorporate different kinds of solvers (in a manner similar to [38]) for handling attribute constraints in the presence of a graph solver that handles partial models. We implemented our approach in the Viatra Solver framework [53]. The source code of our approach is publicly available (https://github.com/viatra/VIATRA-Generator).

We prepared a publicly available measurement environment (https://doi.org/10.5281/zenodo.3950552), and we carried out a detailed experimental evaluation of our approach in three complex case studies to assess scalability and diversity. We obtained favorable scalability results by consistently deriving models with over 250 objects in two cases within an hour, and models with over 1000 objects in the third case with same time limits. These model sizes are substantially larger than logic solver based model generation approaches (e.g. Alloy or Z3) could derive in the presence of structural constraints (see [4, 56, 64]). Moreover, our approach maintains other favorable quality attributes such as diversity and completeness investigated in depth in [55, 69].

**ACKNOWLEDGEMENTS**

We would like to thank Ghanem Soltana and Lionel C. Briand for their help in running the Taxation case study.

This paper is partially supported by MTA-BME Lendület Research Group on Cyber-Physical Systems, BME-Artificial Intelligence FKFP grant of EMMI (BME FKFP-MI/SC), the NSERC RGPIN-04573-16 project, the Fonds de recherche du Québec - Nature et technologies (FRQNT) B1X scholarship (file number: 272709), and Natural Sciences and Engineering Research Council of Canada (NSERC) PGSD3-546810-2020.
REFERENCES


Automated Generation of Consistent Models with Structural and Attribute Constraints


