Symbolic Model Generation for Graph Properties

Sven Schneider, Leen Lambers, Fernando Orejas

Technische Berichte Nr. 115

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam | 115

Sven Schneider | Leen Lambers | Fernando Orejas

Symbolic Model Generation for Graph Properties

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2017
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Druck: docupoint GmbH Magdeburg

ISBN 978-3-86956-396-1

Zugleich online veröffentlicht auf dem Publikationsserver der Universität Potsdam:
URN urn:nbn:de:kobv:517-opus4-103171
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-103171

Graphs are ubiquitous in Computer Science. For this reason, in many areas, it is very important to have the means to express and reason about graph properties. In particular, we want to be able to check automatically if a given graph property is satisfiable. Actually, in most application scenarios it is desirable to be able to explore graphs satisfying the graph property if they exist or even to get a complete and compact overview of the graphs satisfying the graph property.

We show that the tableau-based reasoning method for graph properties as introduced by Lambers and Orejas paves the way for a symbolic model generation algorithm for graph properties. Graph properties are formulated in a dedicated logic making use of graphs and graph morphisms, which is equivalent to first-order logic on graphs as introduced by Courcelle. Our parallelizable algorithm gradually generates a finite set of so-called symbolic models, where each symbolic model describes a set of finite graphs (i.e., finite models) satisfying the graph property. The set of symbolic models jointly describes all finite models for the graph property (complete) and does not describe any finite graph violating the graph property (sound). Moreover, no symbolic model is already covered by another one (compact). Finally, the algorithm is able to generate from each symbolic model a minimal finite model immediately and allows for an exploration of further finite models. The algorithm is implemented in the new tool AutoGraph.

Contents

1 Introduction 8

2 Related Work 10

3 Preliminaries 12

4 Symbolic Model Generation 20

4.1 Sets of symbolic models 20

4.2 Symbolic model generation algorithm $\mathcal{A}$ 21

4.3 Generation of $\mathcal{S}_{NT,k}$ 22

4.4 Compaction of $\mathcal{S}_{NT,k}$ into $\mathcal{S}$ 24

4.5 Explorability of $\mathcal{S}$ 25

5 Implementation 28

6 Conclusion and Outlook 31

A Example Based Comparison with Alloy 37

B Proofs 42

1 Introduction

Graphs are ubiquitous in Computer Science. For this reason, in many areas, it is (or it may be) very important to have the means to express and reason about graph properties. Examples may be, (a) model-based engineering where we may need to express properties of graphical models; (b) the verification of systems whose states are modeled as graphs; (c) to express properties about sets of semi-structured documents, especially if they are related by links; (d) graph databases, where we may want to state integrity constraints in the form of graph properties or where we may want to be able to reason about the validity of graph queries and, in particular, to understand why queries might be valid or not.

Let us take a closer look at the latter application field to understand how the symbolic model generation approach for graph properties, as presented in this paper, will support a *typical usage scenario*. In general, a graph query for a graph database G (as formalized in [3] and used in extended form in [21]) formulates the search for occurrences of graph patterns of a specific form L satisfying some additional property in G. Since such a query can become quite complex it is important to have an intuitive query language to formulate it and to have additional support allowing for reasoning about the query to enhance understandability and facilitate debugging. *Validity* of a graph query means that there should exist a graph database G in which we find an occurrence of the pattern L satisfying the additional property for L encoded in the query, see for example Figure 3.1b depicting a graph property p_1 expressing validity for a query taken from [9, 38] explained in detail in chapter 3. First of all automatic support to answer this validity question for a query is thus desired. Moreover, if validity is the case, then one wants to be able to inspect a graph database G as a *concrete example*, but this example should be of a *manageable size*. Moreover, if there are considerably different types of graph databases being witnessed for the validity of a query then we would like to get a *finite, complete, and compact overview* $\mathcal{S}$ of all these graph databases. Also a *flexible exploration* starting from some minimal example graph database to a bigger one still being a witness for validity is desirable. Finally, of course one wants to see all these results within a *reasonable amount of time*.

For a given graph property p, formulating more generically all requirements occurring in this usage scenario means that we would like to have an algorithm $\mathcal{A}$ returning for p a *finite set of so-called symbolic models* $\mathcal{S}$ such that

- $\mathcal{S}$ jointly covers each finite graph G satisfying p (*complete*),
- $\mathcal{S}$ does not cover any finite graph G violating p (*sound*),
- $\mathcal{S}$ contains no superfluous symbolic model (*compact*),

- $\mathcal{S}$ allows for each of its symbolic models the immediate extraction of a minimal finite graph G covered (*minimally representable*), and
- $\mathcal{S}$ allows an enumeration of further finite graphs G satisfying p (*explorable*).

The contribution of this paper is the presentation and implementation of a *parallelizable* symbolic model generation algorithm delivering a complete (provided termination), sound, compact, minimally representable, and explorable set of symbolic models. We illustrate the algorithm w.r.t. checking validity of some complex graph queries from [9, 38]. Our algorithm takes as input graph properties formulated in an intuitive, dedicated logic making use of graphs and graph morphisms as first-class citizens. This *logic of so-called nested graph conditions* was defined by Habel and Pennemann [14]. A similar approach was first introduced by Rensink [33]. The origins can be found in the notion of graph constraint [16], introduced in the area of graph transformation [34], in connection with the notion of (negative) application conditions [7, 13], as a form to limit the applicability of transformation rules. These graph constraints originally had a very limited expressive power, while nested conditions have been shown [14, 28] to have the *same expressive power as first-order logic (FOL) on graphs* as introduced by Courcelle [4]. Note that because we support FOL on graphs our algorithm might in general not terminate. It is designed however (also if non-terminating) to gradually deliver better underapproximations of the complete set of symbolic models.

This report is an extended version of the paper [36] and is structured as follows: In chapter 2 we give an overview over related work. In chapter 3 we introduce our running example and we reintroduce the key notions of the tableau-based reasoning method that our symbolic model generation algorithm is based on. In chapter 4 we present our algorithm and its formalization and in particular show that it fulfills all requirements. In chapter 5 we describe the algorithm implementation in the new tool AutoGraph. We conclude the paper in chapter 6 together with an overview of future work. In Appendix A we compare based on an example the results obtained from AutoGraph with the results when following the translation based approach facilitating Alloy. Finally, proofs for the included results are contained in Appendix B.

2 Related Work

Instead of using a dedicated logic for graph properties, one can define and reason about graph properties in terms of some existing logic and reuse its associated reasoning methods. In particular, Courcelle [4] studied systematically a graph logic defined in terms of first-order (or monadic second-order) logic. In that approach, graphs are defined axiomatically using predicates $node(n)$, asserting that n is a node, and $edge(n_1, n_2)$ asserting that there is an edge from n_1 to n_2. Such a *translation-based approach* for finding models of graph-like properties is followed, for example, in [11], where OCL properties are translated into relational logic, and reasoning is then performed by KODKOD, a SAT-based constraint solver for relational logic. In a similar vein, in [1] reasoning for feature models is being provided based on a translation into input for different general-purpose reasoners. Analogously, in [37] the ALLOY analyzer is used to synthesize in this case large, well-formed and realistic models for domain-specific languages. Reasoning for domain specific modeling is addressed also in [19, 20] using the FORMULA approach taking care of dispatching the reasoning to the state-of-the-art SMT solver Z3. In [35] another translation-based approach is presented to reason with so-called partial models expressing uncertainty about the information in the model during model-based software development. In principle, all the previously exemplarily presented approaches from the model-based engineering domain represent potential use cases for our dedicated symbolic model generation approach for graph-like properties. Since we are able to generate *symbolic models* being complete (in case of termination), sound, compact, minimally representable, and explorable in combination, we believe that our approach has the potential to enhance considerably the type of analysis results, in comparison with the results obtained by using off-the-shelf SAT-solving technologies.

Following this idea, in contrast to the translation-based approach it is possible, e.g, to formalize a graph-like property language such as OCL [32] by a dedicated logic for graph properties [14] and apply corresponding *dedicated automated reasoning methods* as developed in [23, 26, 27, 29]. The advantage of such a graph-dedicated approach as followed in this paper is that graph axioms are natively encoded in the reasoning mechanisms of the underlying algorithms and tooling. Therefore, they can be built to be more efficient than generic-purpose methods as demonstrated for example in [27, 28, 29], where such an approach outperforms some standard provers working over encoded graph conditions. Moreover, the translation effort for each graph property language variant (such as for example OCL) into a formal logic already dedicated to the graph domain is much smaller than a translation into some more generic logic, which in particular makes translation errors less probable. As most directly related work [27, 28] presents a satisfiability solving algorithm for graph properties as employed in this paper [14]. This solver attempts to find one finite model (if possible), but does not generate a compact and gradually complete finite set of symbolic models allowing to inspect all possible finite models including a finite set of minimal ones. In contrast to [27, 28] our symbolic model generation algorithm is interleaved directly with a refutationally complete tableau-based reasoning method [23], inspired by rules of a proof system presented previously in [29], but in that work the proof rules were not shown to be refutationally complete.

3 Preliminaries

In this section we first introduce our running example and then recall definitions and results from [23] simplified for their application in subsequent sections.

We consider as an example two social network queries as described in the Social Network Benchmark developed by the Linked Data Benchmark Council [9, 38]. The form of social networks to be queried is given by the type graph in Figure 3.1a. Moreover, we forbid parallel edges of the same type. The first considered graph query (a variant of query 8 from [3]) looks for pairs of *Person*s and *Tag*s such that in such a pair a *Tag* is new in some *Post* by a friend of this *Person*. To be a *Post* of a friend, the *Post* must be from a second *Person* the *Person knows*. In order to be new, the *Tag* must be linked in the latest *Post* of the second *Person* (and thus in a *Post* that has no *successor Post*) and there has to be no former *Post* by any other or the same friend that is not her last one and where the same *Tag* has been already used. In both cases only *Tag*s that are not simply inherited from a *link*ed *Post* should be considered. This query is valid if there is a graph database G in which such a *Person* and *Tag* pair can be found at least once. The corresponding graph property p_1 is depicted in Figure 3.1b. The graph property p_2 for a variant of query 10 [9, 38] is given in Figure 3.1c.

Technically, we express graph properties as a special case of nested graph conditions that are formulated over a graph C and satisfied by monomorphisms (monos for short) [14]. In particular, a graph property satisfied by a graph G is a graph condition over the empty graph $\emptyset$ satisfied by the unique mono $\emptyset \to G$.

Definition 1 (condition, property) *We define conditions over graphs inductively:*

- $\exists(m, c)$ *is a condition over* C*, if* $m : C \hookrightarrow D$ *is a mono and* c *is a condition over* D*,*
- $\neg c$ *is a condition over* C*, if* c *is a condition over* C*, and*
- $\wedge(c_1, \ldots, c_k)$ *is a condition over* C*, if* $c_1, \ldots, c_k$ *are conditions over* C*.*

A graph property *is a condition over the empty graph* $\emptyset$*.*

Note, the empty conjunction $\wedge()$ serves as a base case for the inductive definition. Without extending expressiveness of the conditions, we define the following operators: $\vee(c_1, \ldots, c_k) := \neg \wedge (\neg c_1, \ldots, \neg c_k)$, *true*$:=\wedge()$, *false*$:=\vee()$, and $\forall(m, c) := \neg\exists(m, \neg c)$. Finally, we also use $\wedge(S)$ if S is a finite set instead of a list.

Definition 2 (satisfaction) *A mono* $q : C \hookrightarrow G$ *satisfies a condition* c *over* C*, written,* $q \models c$ *as follows.*

- *if* $c = \exists(m : C \hookrightarrow D, c')$ *and there is a mono* $q' : D \hookrightarrow G$ *such that* $q' \circ m = q$ *and* $q' \models c$*, then* $q \models c$*.*

- *if $c = \neg c'$ and $q \not\models c'$, then $q \models c$.*
- *if $c = \wedge(c_1, \dots, c_k)$, $q \models c_1, \dots, q \models c_k$, then $q \models c$.*

Finally, if G is a graph, p is a graph property, and the unique mono $i : \emptyset \hookrightarrow G$ satisfies p, then G satisfies p, written $G \models p$.

For example, the graph $\circ\!\circlearrowleft$ satisfies the condition $c = \exists(m_1 : \emptyset \hookrightarrow \circ, \neg\exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true}))$ as explained by the following diagram. According to Definition 2 $\circ\!\circlearrowleft \models c$ if for the unique inclusion $i : \emptyset \hookrightarrow \circ\!\circlearrowleft$ satisfies c. Then, $i : \emptyset \hookrightarrow \circ\!\circlearrowleft \models c$ if there is some $p_1 : \circ \hookrightarrow \circ\!\circlearrowleft$ such that $p_1 \circ m_1 = i$ and $p_1 \models \neg\exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true})$. Obviously, there is only one possible p_1, which also satisfies $p_1 \circ m_1 = i$. Then, $p_1 \models \neg\exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true})$ if we can derive a contradiction from $p_1 \models \exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true})$. $p_1 \models \exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true})$ implies that there is some $p_2 : \circ\,\circ \hookrightarrow \circ\!\circlearrowleft$ such that $p_2 \circ m_2 = p_1$ and $p_2 \models \mathit{true}$. However, this is a contradiction because no such mono p_2 exists. This contradiction implies that $p_1 \models \neg\exists(m_2 : \circ \hookrightarrow \circ\,\circ, \mathit{true})$ as desired.

$$\begin{tikzcd}
\emptyset \arrow[r, hook, "m_1"] \arrow[dr, hook, "i"'] & \circ \arrow[r, hook, "m_2"] \arrow[d, hook, "p_1"] & \circ\,\circ \arrow[dl, hook, "p_2\ \lightning"] \\
 & \circ\!\circlearrowleft &
\end{tikzcd}$$

Note that we reintroduced the definitions above for graphs, but our results can be generalized to variants of graphs such as, for example, typed attributed graphs, Petri nets, or even algebraic specifications, since they belong to an $\mathcal{M}$-adhesive category [6, 22] satisfying some additional categorical properties that the tableau-based reasoning method [23] requires. This is another advantage as opposed to using encodings as referred to in related work, since each kind of graph structure would otherwise need a different encoding. Moreover, since many DB queries typically make use of attributes by using variables, we will extend our algorithm to typed *attributed* graphs in the future: then, the monos from the answer set also determine an assignment of variables to values from the database. At the current stage of development such further attribute constraints are not checked and, therefore, monos would be returned for which such attribute constraints would have to be checked a posteriori. Finally, it may also be desireable to provide automatic translations for (a well-defined subset of) graph DB query languages to AutoGraph. Such translations could then enable the usage of AutoGraph as a backend reasoner for graph queries replacing translations into more generic FOL reasoners (as for the model-based engineering scenario in [21]).

Our symbolic model generation method will operate on the subset of conditions in conjunctive normal form (CNF), simplifying the corresponding reasoning. For example, $\wedge(\vee()) = \wedge(\mathit{false})$ is a condition in CNF equivalent to *false*. We therefore assume an operation $[\cdot]$, similarly to operations in [23, 28, 29], translating conditions into equivalent conditions in CNF. This operation applies, besides the expected equivalences, like the equivalence for removal of universal quantification mentioned before Definition 2, an equivalence for the removal of literals with isomorphisms (for example, $\exists(i : A \xrightarrow{\sim} B, \exists(m : B \hookrightarrow C, true))$ is replaced by $\exists((m \circ i) : A \hookrightarrow C, true)$ by moving the isomorphism i into the literals of the next nesting level). See Figure 3.2 for a complete list. In particular, a negative literal in CNF is trivially satisfiable by the identity morphism, a property that will be exploited heavily in our symbolic model generation algorithm. Note, skolemization, which removes existential quantification in FOL SAT-reasoning, is not needed for graph conditions [28, p. 100]; we employ CNF-conversion on quantified subconditions separately.

Definition 3 (CNF) *A* literal ℓ *is either a* positive literal $\exists(m, c)$ *or a* negative literal $\neg\exists(m, c)$ *where m is no isomorphism and c is in CNF. A* clause *is a disjunction of literals. A conjunction of clauses is a condition in* CNF.

Remark 1 (Runtime Complexity of Conversion to CNF) *As for FOL the conversion to CNF entails the conversion of subconditions of the shape* $(a_1 \wedge b_1) \vee \cdots \vee (a_n \wedge b_n)$ *resulting in* 2^n *clauses of size n. However, in our approach the conversion of graph conditions into CNF graph conditions usually has no great impact because subconditions from different existential quantifiers are not combined in the conversion, that is, we perform the conversion on each nesting level of the* $\exists$*-quantifier and, hence, we obtain quite small CNF formulas as inputs. For FOL this is different: after skolemization, which removes existential quantifiers, all subconditions are related to each other resulting in huge formulas.*

In the running example the conversion to CNF requires negligable time (up to 4 ms) and increases the number of operators from 45 to 61 for the condition $p_1 \wedge p_2$ *from Figure 3.1. For a comparison, consider the FOL enconding of this condition in* Alloy *from Appendix A. When using* Alloy *to perform the CNF conversion we obtain 199 122 clauses in 2169 ms.*

Also note, the size of the graphs and the morphisms contained in the condition are not relevant for the conversion in our case, which is an important difference to the FOL scenario in Alloy.

The tableau-based reasoning method as introduced in [23] is based on so-called nested tableaux. We start with reintroducing the notion of a regular tableau for a graph condition, which was directly inspired by the construction of tableaux for plain FOL reasoning [15]. Intuitively, provided a condition in CNF, such an iteratively constructed tableau represents all possible selections (computed using the extension rule in the following definition) of precisely one literal from each clause of the condition (note, a condition is unsatisfiable if it contains an empty clause). Such a selection is given by a maximal path in the tableau, which is called branch. In this sense, we are constructing a disjunctive normal form (DNF) where the set of nodes occurring in a branch of the resulting tableau corresponds to one clause of this DNF. Then, to discover contradictions in the literals of a branch and to prepare for the next step in the satisfiability analysis we merge (using the lift rule in the following definition) the selected literals into a single positive literal (note, if no positive literal is available the condition is always satisfiable), which is called opener. Note that the lift rule is based on a shifting translating a condition over a morphism into an equivalent condition [8, 14].

Definition 4 (tableau, tableau rules, open/closed branches) *Given a condition c in CNF over C. A* tableau *T for c is a tree whose nodes are conditions constructed using the rules below. A* branch *in a tableau T for c is a maximal path in T. Moreover, a branch is* closed *if it contains false; otherwise, it is* open*. Finally, a tableau is* closed *if all of its branches are closed; otherwise, it is* open*.*

- initialization rule*:*
  a tree with a single root node true is a tableau for c.
- extension rule*:*
  if T is a tableau for c, B is a branch of T, and $\vee(c_1, \ldots, c_n)$ is a clause in c, then if $n > 0$ and $c_1, \ldots, c_n$ are not in B, then extend B with n child nodes $c_1, \ldots, c_n$ or if $n = 0$ and false is not in B, then extend B with false.
- lift rule*:*
  if T is a tableau for c, B is a branch of T, $\exists(m, c')$ and ℓ are literals in B, $\ell' = \exists(m, [c' \wedge \mathit{shift}(m, \ell)])$ is not in B, then extend B with ℓ'.

The operation $\mathit{shift}(\cdot,\cdot)$ allows to shift conditions over morphisms preserving satisfaction in the sense that $m_1 \circ m_2 \models c$ iff $m_1 \models \mathit{shift}(m_2, c)$ (see [23, lemma 3]). Semi-saturated tableaux are the desired results of the iterative construction where no further rules need to be applied.

Definition 5 (semi-saturation, hook of a branch) *Let T be a tableau for condition c over C. A branch B of T is* semi-saturated *if it is either closed or*

- *B is not extendable with a new node using the extension rule and*
- *if $E = \{\ell_1, \ldots, \ell_n\}$ is nonempty and the set of literals added to B using the extension rule, then there is a positive literal $\ell = \exists(m, c')$ in E such that the literal in the leaf of B is equivalent to $\exists(m, c' \wedge_{\ell' \in (E - \{\ell\})} \mathit{shift}(m, \ell'))$. Also, we call ℓ the* hook *of B.*

Finally, T is semi-saturated *if all its branches are semi-saturated.*

In fact, a condition c is satisfiable if and only if the leaf condition of some open branch of a corresponding semi-saturated tableau is satisfiable. Hence, the next analysis step is required if there is a leaf $\exists(m : C \hookrightarrow C', c')$ of some open branch for which satisfiability has to be decided. That is, the next analysis step is to construct a tableau for condition c'. The iterative (possibly non-terminating) execution of this procedure results in (possibly infinitely many) tableaux where each tableau may result in the construction of a finite number of further tableaux. This relationship between a tableau and the tableaux derived from the leaf literals of open branches results in a so called nested tableau (see Figure 3.3 for an example of a nested tableau).

Definition 6 (nested tableau, opener, context, nested branch, semi-saturation)
Given a condition c over C and a poset $(I, \leq, i_0)$ with minimal element i_0. A nested tableau *NT for c is for some $I' \subseteq I$ a family of triples $\{\langle T_i, j, c_i \rangle\}_{i \in I'}$ constructed using the following rules.*

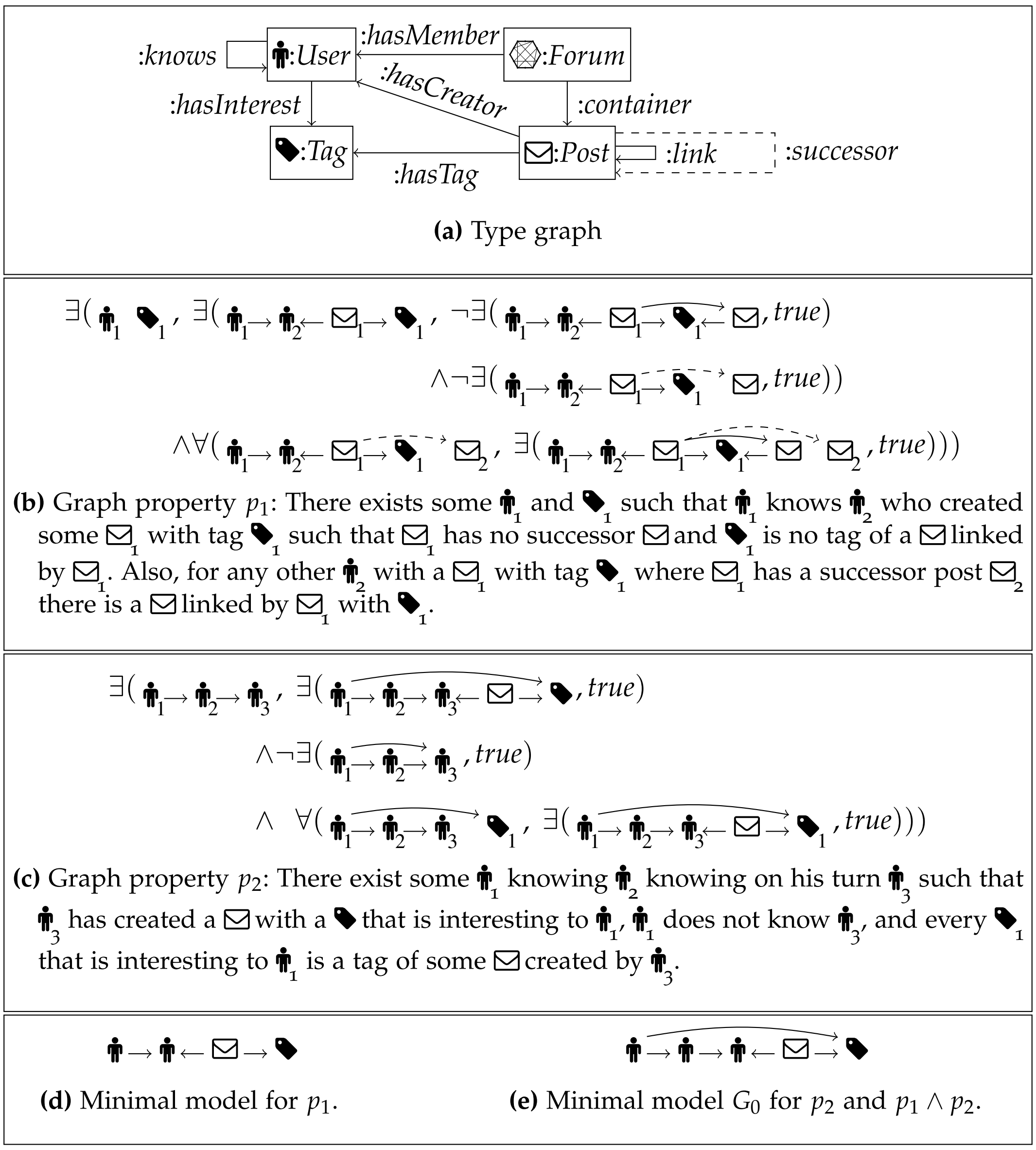
- initialization rule*:*
  If T_{i_1} is a tableau for c, then the family containing only $\langle T_{i_1}, i_0, true \rangle$ for some index $i_1 > i_0$ is a nested tableau for c and C is called context *of T_{i_1}.*

- nesting rule*:*
  If NT is a nested tableau for c with index set I', $\langle T_n, k, c_k \rangle$ is in NT for index n, the literal $\ell = \exists(m_n : A_n \hookrightarrow A_j, c_n)$ is a leaf of T_n, ℓ is not the condition in any other triple of NT, T_j is a tableau for c_n, and $j > n$ is some index not in I', then add the triple $\langle T_j, n, \ell \rangle$ to NT using index j, ℓ is called opener *of T_j, and A_j is called* context *of T_j.*

A nested branch *NB of the nested tableau NT is a maximal sequence of branches $B_{i_1}, \ldots, B_{i_k}, B_{i_{k+1}}, \ldots$ of tableaux $T_{i_1}, \ldots, T_{i_k}, T_{i_{k+1}}, \ldots$ in NT starting with a branch B_{i_1} in the initial tableau T_{i_1} of NT, such that if B_{i_k} and $B_{i_{k+1}}$ are consecutive branches in the sequence then the leaf of B_{i_k} is the opener of $T_{i_{k+1}}$. NB is* closed *if it contains a closed branch; otherwise, it is* open. *NT is* closed *if all its nested branches are closed. Finally, NT is* semi-saturated *if each tableau in NT is semi-saturated.*

It has been shown in [23] that the tableau based reasoning method using nested tableaux for conditions c is sound and refutationally complete. In particular, soundness means that if we are able to construct a nested tableau where all its branches are closed then the original condition c is unsatisfiable. Refutational completeness means that if a *saturated* tableau includes an open branch, then the original condition is satisfiable. In fact, each open finite or infinite branch in such a tableau defines a finite or infinite model of the property, respectively. Informally, the notion of saturation requires that all tableaux of the given nested tableau are semi-saturated and that hooks are selected in a *fair* way not postponing indefinitely the influence of a positive literal for detecting inconsistencies leading to closed nested branches.

Remark 2 (Refutational Completeness and Fairness)
Incompleteness can be caused in tableaux for FOL by unfair selection of formulas (confer [15, page 117, Figure 4] for an example where the unsatisfiable condition $Q \land \neg Q$ is treated unfair by never being selected). In our case the set of conditions from which a hook is to be selected in a fair way changes from one tableau to the next because conditions that are not selected are lifted into the hook resulting (possibly) in various conditions. These conditions are called the successors of the not selected, lifted conditions in [23]. To ensure refutational completeness we ensure that the impact of a condition affects the nested branch eventually by not postponing the selection of one these successors as a hook indefinitely. Confer to [23, p. 29] for the discussion in the original paper.



(a) Type graph

(b) Graph property p_1: There exists some 👤$_1$ and 🏷$_1$ such that 👤$_1$ knows 👤$_2$ who created some ✉$_1$ with tag 🏷$_1$ such that ✉$_1$ has no successor ✉ and 🏷$_1$ is no tag of a ✉ linked by ✉$_1$. Also, for any other 👤$_2$ with a ✉$_1$ with tag 🏷$_1$ where ✉$_1$ has a successor post ✉$_2$ there is a ✉ linked by ✉$_1$ with 🏷$_1$.

(c) Graph property p_2: There exist some 👤$_1$ knowing 👤$_2$ knowing on his turn 👤$_3$ such that 👤$_3$ has created a ✉ with a 🏷 that is interesting to 👤$_1$, 👤$_1$ does not know 👤$_3$, and every 🏷$_1$ that is interesting to 👤$_1$ is a tag of some ✉ created by 👤$_3$.

(d) Minimal model for p_1.

(e) Minimal model G_0 for p_2 and $p_1 \wedge p_2$.

Figure 3.1: Graph properties for queries from the Social Network Benchmark [9, 38]

unfold abbreviations except for disjunction

$$\begin{aligned} \mathit{true} &\equiv \wedge() \\ \mathit{false} &\equiv \vee() \\ \forall(m,c) &\equiv \neg\exists(m,\neg c) \end{aligned}$$

move negation in front of existential quantification

$$\begin{aligned} \neg \wedge (S) &\equiv \vee(\{\neg c \mid c \in S\}) \\ \neg \vee (c_1,\dots,c_n) &\equiv \wedge(\{\neg c \mid c \in S\}) \\ \neg\neg c &\equiv c \end{aligned}$$

ensure that a conjunction contains only disjunctions and literals

$$\wedge(S \cup \{\wedge(S')\}) \equiv \wedge(S \cup S')$$

ensure that a disjunction contains only conjunctions and literals

$$\vee(S \cup \{\vee(S')\}) \equiv \vee(S \cup S')$$

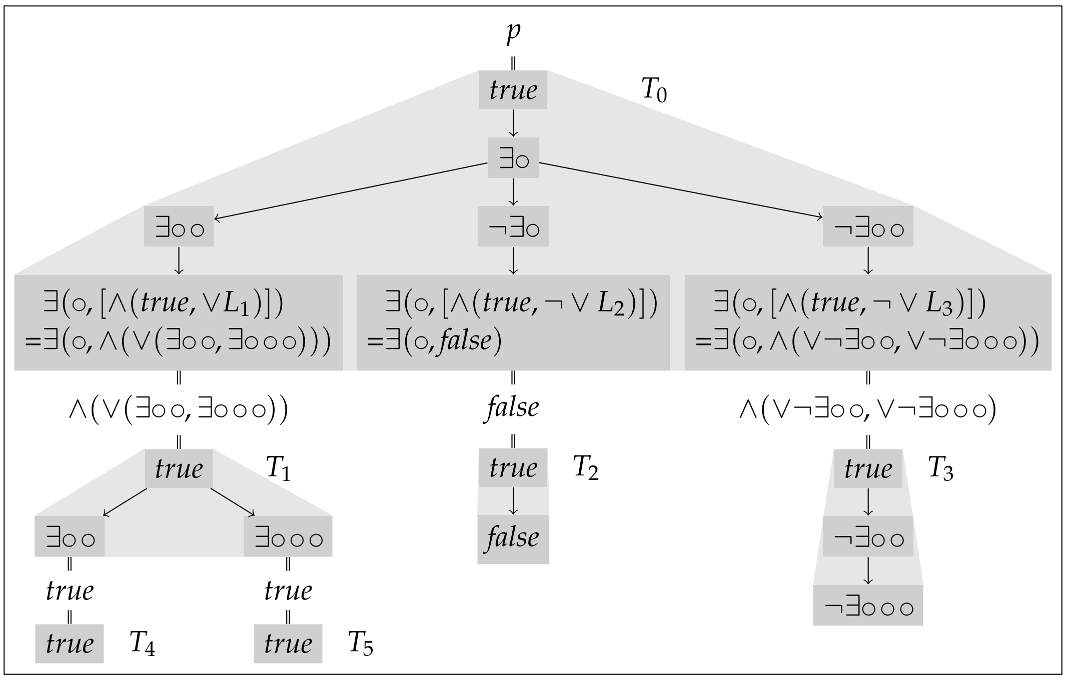
ensure that a conjunction contains only disjunctions of literals

$$\wedge(S \cup \{\vee(S' \cup \{\wedge(S'')\})\}) \equiv \wedge(S \cup \{\vee(S' \cup \{s''\}) \mid s'' \in S''\})$$

ensure absence of isomorphisms

$$\begin{aligned} \exists(m : A \xrightarrow{\sim} A, c) &\equiv \text{move-down}(m,c) \\ \text{move-down}(m,\wedge(S)) &\equiv \wedge(\{\text{move-down}(m,s) \mid s \in S\}) \\ \text{move-down}(m,\vee(S)) &\equiv \vee(\{\text{move-down}(m,s) \mid s \in S\}) \\ \text{move-down}(m,\neg c) &\equiv \neg\text{move-down}(m,c) \\ \text{move-down}(m,\exists(m',c)) &\equiv \exists(m' \circ m, c) \end{aligned}$$

Figure 3.2: The conversion of conditions into CNF, written $[\cdot]$, applies the equivalences above from left to right



p
$true$
T_0
$\exists\circ$
$\exists\circ\circ$
$\neg\exists\circ$
$\neg\exists\circ\circ$
$\exists(\circ,[\wedge(true,\vee L_1)])$
$=\exists(\circ,\wedge(\vee(\exists\circ\circ,\exists\circ\circ\circ)))$
$\exists(\circ,[\wedge(true,\neg\vee L_2)])$
$=\exists(\circ,false)$
$\exists(\circ,[\wedge(true,\neg\vee L_3)])$
$=\exists(\circ,\wedge(\vee\neg\exists\circ\circ,\vee\neg\exists\circ\circ\circ))$
$\wedge(\vee(\exists\circ\circ,\exists\circ\circ\circ))$
$false$
$\wedge(\vee\neg\exists\circ\circ,\vee\neg\exists\circ\circ\circ)$
$true$ T_1
$true$ T_2
$true$ T_3
$\exists\circ\circ$
$\exists\circ\circ\circ$
$false$
$\neg\exists\circ\circ$
$\neg\exists\circ\circ\circ$
$true$
$true$
$true$ T_4
$true$ T_5

Figure 3.3: Nested tableau (consisting of tableau $T_0, \ldots, T_5$) for graph property $p = \exists(\circ, true) \wedge (\vee(\exists(\circ\circ, true), \neg\exists(\circ, true), \neg\exists(\circ\circ, true)))$. In the middle branch *false* is obtained because $\neg \vee L_2$ is reduced to *false* because $\vee L_2$ is reduced to *true* because L_2 contains $\exists\circ$ due to shifting, which is reduced by $[\cdot]$ to *true* because of the used isomorphism. We extract from the nested branches ending in T_4, T_5, and T_3 the symbolic models $\langle\circ\circ, true\rangle$, $\langle\circ\circ\circ, true\rangle$, and $\langle\circ, \wedge(\neg\exists\circ\circ, \neg\exists\circ\circ\circ)\rangle$. Here $\langle\circ\circ\circ, true\rangle$ is a refinement of $\langle\circ\circ, true\rangle$ and, hence, would be removed by compaction as explained in section 4.4

4 Symbolic Model Generation

In this section we present our symbolic model generation algorithm. We first formalize the requirements from the introduction for the generated set of symbolic models, then present our algorithm, and subsequently verify that it indeed adheres to these formalized requirements. In particular,we want our algorithm to extract symbolic models from all open finite branches in a saturated nested tableau constructed for a graph property p. This would be relatively straightforward if each saturated nested tableau would be finite.

However, in general, as stated already at the end of the previous section this may not be the case. E.g., consider the conjunction $p_0 = \wedge(p_1, p_2, p_3)$ of the conditions $p_1 = \exists($①$, \forall($◯→①$, \mathit{false}))$ (there is a node which has no predecessor), $p_2 = \forall($①$, \exists($①→◯$, \mathit{true}))$ (every node has a successor), and $p_3 = \forall($◯→◯←◯$, \mathit{false})$ (no node has two predecessors), which is only satisfied by the infinite graph $G_\infty =$ ◯→◯→◯→◯→ $\cdots$.

Thus, in order to be able to find a complete set of symbolic models without knowing beforehand if the construction of a saturated nested tableau terminates, we introduce the key-notions of k-semi-saturation and k-termination to reason about nested tableaux up to depth k, which are in some sense a prefix of a saturated tableau. Note, the verification of our algorithm, in particular for completeness, is accordingly based on induction on k. Informally, this means that by enlarging the depth k during the construction of a saturated nested tableau, we eventually find all finite open branches and thus finite models. This procedure will at the same time guarantee that we will be able to extract symbolic models from finite open branches even for the case of an infinite saturated nested tableau. E.g., we will be able to extract $\emptyset$ from a finite open branch of the infinite saturated nested tableau for property $p_4 = \wedge(p_1 \vee \exists($①$, \mathit{false}), p_2, p_3)$.

4.1 Sets of symbolic models

The symbolic model generation algorithm $\mathcal{A}$ should generate for each graph property p a set of symbolic models $\mathcal{S}$ satisfying all requirements described in the introduction. A symbolic model in its most general form is a graph condition over a graph C, where C is available as an explicit component. A symbolic model then represents a possibly empty set of graphs (as defined subsequently in Definition 10). A specific set of symbolic models $\mathcal{S}$ for a graph property p satisfies the requirements soundness, completeness, minimal representability, and compactness if it adheres to the subsequent formalizations of these notions.

Definition 7 (symbolic model) *If c is a condition over C according to Definition 1, then $\langle C, c\rangle$ is a symbolic model.*

Based on the notion of m-consequence we relate symbolic models subsequently.

Definition 8 (m-consequence on conditions) *If c_1 and c_2 are conditions over C_1 and C_2, respectively, $m : C_1 \hookrightarrow C_2$ is a mono, and for all monos $m_1 : C_1 \hookrightarrow G$ and $m_2 : C_2 \hookrightarrow G$ such that $m_2 \circ m = m_1$ it holds that $m_2 \models c_2$ implies $m_1 \models c_1$, then c_1 is an m-consequence of c_2, written $c_2 \vdash_m c_1$. We can state the existence of such an m by writing $c_2 \vdash c_1$. We also omit m if it is the identity or clear from the context. Finally, conditions c_1 and c_2 over C are equivalent, written $c_1 \equiv c_2$, if $c_1 \vdash c_2$ and $c_2 \vdash c_1$.*

We define coverage of symbolic models based on the notion of m-refinement, which relies on an m-consequence between the contained conditions.

Definition 9 (m-refinement of symbolic model) *If $\langle C_1, c_1\rangle$ and $\langle C_2, c_2\rangle$ are symbolic models and $m : C_1 \hookrightarrow C_2$ is a mono, and $c_2 \vdash_m c_1$, then $\langle C_2, c_2\rangle$ is an m-*refinement *of $\langle C_1, c_1\rangle$, written $\langle C_2, c_2\rangle \leq_m \langle C_1, c_1\rangle$. The set of all such symbolic models $\langle C_2, c_2\rangle$ is denoted by refined$(\langle C_1, c_1\rangle)$.*

We define the graphs covered by a symbolic model as follows.

Definition 10 (m-covered by a symbolic model) *If $\langle C, c\rangle$ is a symbolic model, G is a finite graph, $m : C \hookrightarrow G$ is a mono, and $m \models c$ then G is an m-*covered graph *of $\langle C, c\rangle$. The set of all such graphs is denoted by covered$(\langle C, c\rangle)$. For a set $\mathcal{S}$ of symbolic models covered$(\mathcal{S}) = \cup_{s \in S}$covered$(s)$.*

Based on these definitions, we formalize the first four requirements from chapter 1 to be satisfied by the sets of symbolic models returned by algorithm $\mathcal{A}$.

Definition 11 (sound, complete, minimally representable, compact) *Given a set $\mathcal{S}$ of symbolic models and a graph property p. $\mathcal{S}$ is* sound *w.r.t. p if covered$(\mathcal{S}) \subseteq \{G \mid G \models p \land G$ is finite$\}$, $\mathcal{S}$ is* complete *w.r.t. p if covered$(\mathcal{S}) \supseteq \{G \mid G \models p \land G$ is finite$\}$, $\mathcal{S}$ is* minimally representable *w.r.t. p if for each $\langle C, c\rangle \in \mathcal{S}$: $C \models p$ and for each $G \in$ covered$(\langle C, c\rangle)$ there is a mono $m : C \hookrightarrow G$, and $\mathcal{S}$ is* compact *if all $(s_1 \neq s_2) \in \mathcal{S}$ satisfy covered$(s_1) \not\subseteq$ covered(s_2).*

4.2 Symbolic model generation algorithm $\mathcal{A}$

We briefly describe the two steps of the algorithm $\mathcal{A}$, which generates for a graph property p a set of symbolic models $\mathcal{A}(p) = \mathcal{S}$. The algorithm consists of two steps: the generation of symbolic models and the compaction of symbolic models, which are discussed in detail in section 4.3 and section 4.4, respectively. Afterwards, in section 4.5, we discuss the explorability of the obtained set of symbolic models $\mathcal{S}$.

Step 1 (Generation of symbolic models in section 4.3). We apply the tableau and nested tableau rules from chapter 3 to iteratively construct a nested tableau. Then, we extract symbolic models from certain nested branches of this nested tableau that can not be extended. Since the construction of the nested tableau may not terminate due to infinite nested branches we construct the nested tableau in breadth-first manner and extract the symbolic models whenever possible. Moreover, we eliminate a source of nontermination by selecting the hook in each branch in a fair way not postponing the successors of a positive literal that was not chosen as a hook yet indefinitely [23, p. 29] ensuring at the same time refutational completeness of our algorithm. This step ensures that the resulting set of symbolic models is sound, complete (provided termination), and minimally representable. The symbolic models extracted from the intermediately constructed nested tableau NT for growing k is denoted $\mathcal{S}_{NT,k}$.

Step 2 (Compaction of symbolic models in section 4.4). We obtain the final result $\mathcal{S}$ from $\mathcal{S}_{NT,k}$ by the removal of symbolic model that are a refinement of any other symbolic model. This step preserves soundness (as only symbolic models are removed), completeness (as only symbolic models are removed that are refinements, hence, the removal does not change the set of covered graphs), and minimal representability (as only symbolic models are removed), and additionally ensures compactness.

4.3 Generation of $\mathcal{S}_{NT,k}$

By applying a breadth-first construction we build nested tableaux that are for increasing k, both, k-semi-saturated, stating that all branches occurring up to index k in all nested branches are semi-saturated, and k-terminated, stating that no nested tableau rule can be applied to a leaf of a branch occurring up to index k in some nested branch.

Definition 12 (k-semi-saturation, k-terminated) *Given a nested tableau NT for condition c over C. If NB is a nested branch of length k of NT and each branch B contained at index $i \leq k$ in NB is semi-saturated, then NB is* k-semi-saturated. *If every nested branch of NT of length n is $min(n,k)$-semi-saturated, then NT is* k-semi-saturated. *If NB is a nested branch of NT of length n and the nesting rule can not be applied to the leaf of any branch B at index $i \leq min(n,k)$ in NB, then NB is* k-terminated. *If every nested branch of NT of length n is $min(n,k)$-terminated, then NT is* k-terminated. *If NB is a nested branch of NT that is* k-terminated *for each k, then NB is* terminated. *If NT is* k-terminated *for each k, then NT is* terminated.

We define the k'-remainder of a branch, which is a refinement of the condition of that tableau, that is used by the subsequent definition of the set of extracted symbolic models.

Definition 13 (k'-remainder of branch) *Given a tableau T for a condition c over C, a mono $q : C \hookrightarrow G$, a branch B of T, and a prefix P of B of length $k' > 0$. If R contains (1) each condition contained in P unless it has been used in P by the lift rule (being $\exists(m, c')$ or ℓ in the lift rule in Definition 4) and (2) the clauses of c not used by the extension* *rule in P (being $\vee(c_1, \ldots, c_n)$ in the extension rule in Definition 4), then $\langle C, \wedge R\rangle$ is the k'-*remainder *of B.*

The set of symbolic models extracted from a nested branch *NB* is a set of certain k'-remainders of branches of *NB*. In the example given in Figure 3.3 we extracted three symbolic models from the four nested branches of the nested tableau.

Definition 14 (extracted symbolic model) *If NT is a nested tableau for a condition c over C, NB is a k-terminated and k-semi-saturated nested branch of NT of length $n \leq k$, B is the branch at index n of length k' in NB, B is open, B contains no positive literals, then the k'-*remainder *of B is the symbolic model extracted from B. The set of all such extracted symbolic models from k-terminated and k-semi-saturated nested branches of NT is denoted $\mathcal{S}_{NT,k}$.*

Based on the previously introduced definitions of soundness, completeness, and minimal representability of sets of symbolic models w.r.t. graph properties we are now ready to verify the corresponding results on the algorithm $\mathcal{A}$.

Theorem 1 (soundness) *If NT is a nested tableau for a graph property p, then $\mathcal{S}_{NT,k}$ is sound w.r.t. p.*

Theorem 2 (completeness) *If NT is a terminated nested tableau for a graph property p, k is the maximal length of a nested branch in NT, then $\mathcal{S}_{NT,k}$ is complete w.r.t. p.*

As explained by the example at the beginning of chapter 4 the algorithm may not terminate. However, the symbolic models extracted at any point during the construction of the nested tableau are a gradually extended underapproximation of the complete set of symbolic models. Moreover, the openers $\exists(m : G_1 \hookrightarrow G_2, c)$ of the branches that end nonterminated nested branches constitute an overapproximation by encoding a lower bound on missing symbolic models in the sense that each symbolic model that may be discovered by further tableau construction contains some G_2 as a subgraph.

Theorem 3 (minimal representability) *If NT is a nested tableau for a graph property p, then $\mathcal{S}_{NT,k}$ is minimally representable w.r.t p.*

For $p_1 \wedge p_2$ from Figure 3.1 we obtain a terminated nested tableau (consisting of 114 tableaux with 25 032 nodes) from which we generate 28 symbolic models (with a total number of 5433 negative literals in their negative remainders). For p from Figure 3.3 we generate 3 symbolic models, which are given also in Figure 3.3. In the next subsection we explain how to compact sets of symbolic models.

As a further example, consider the univerally quantified graph property $\forall(m : \emptyset \hookrightarrow G, c)$, where G is not the empty graph, which is converted into the CNF graph property $\neg\exists(m, \neg c)$. We can then construct a nested tableau for this graph property, which contains a single tableau with a single branch with the single item $\neg\exists(m, \neg c)$. The smallest graph satisfying $\forall(m, c)$ (or, equivalently, $\neg\exists(m, \neg c)$) is the empty graph $\emptyset$. The algorithm $\mathcal{A}$ reflects this by returning $\{\langle\emptyset, \neg\exists(m, \neg c)\rangle\}$ as the set of generated symbolic models.

4.4 Compaction of $\mathcal{S}_{NT,k}$ into $\mathcal{S}$

The set of symbolic models $\mathcal{S}_{NT,k}$ as obtained in the previous section can be compacted by application of the following lemma. It states a sufficient condition for whether a symbolic model $\langle A_1, c_1 \rangle$ refines another symbolic model $\langle A_2, c_2 \rangle$, which is equivalent to $\mathit{covered}(\langle A_1, c_1 \rangle) \supseteq \mathit{covered}(\langle A_2, c_2 \rangle)$. In this case we can remove the covered symbolic model $\langle A_2, c_2 \rangle$ from $\mathcal{S}_{NT,k}$ without changing the graphs covered. Since the set of symbolic models $\mathcal{S}_{NT,k}$ is always finite we can apply the following lemma until no further coverages are determined.

Lemma 1 (compaction) *If $\langle A_1, c_1 \rangle$ and $\langle A_2, c_2 \rangle$ are symbolic models, $m : A_1 \hookrightarrow A_2$ is a mono, and the condition $\exists(i_2, c_2 \wedge \neg \mathit{shift}(i_2, \exists(i_1, c_1)))$ is not satisfiable by a finite graph G, then $\mathit{covered}(\langle A_1, c_1 \rangle) \supseteq \mathit{covered}(\langle A_2, c_2 \rangle)$.*

We may apply this lemma when we determine a mono m such that the condition $\exists(i_2, c_2 \wedge \neg \mathit{shift}(i_2, \exists(i_1, c_1)))$ is refutable. For this latter part we apply our tableau construction as well and terminate as soon as non-refutability is detected, that is, as soon as a symbolic model is obtained for the condition.

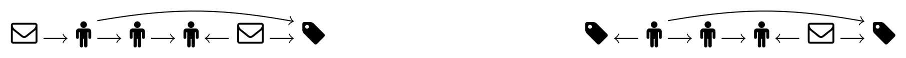
For the resulting set $\mathcal{S}$ of symbolic models obtained from iterated application of Lemma 1 we now state the compactness as defined before.

Theorem 4 (compactness) *If NT is a nested tableau for a graph property p, then $\mathcal{S} \subseteq \mathcal{S}_{NT,k}$ is compact.*

For $p_1 \wedge p_2$ from Figure 3.1 we determined a single symbolic model with minimal model (given in Figure 3.1e) that is contained by the minimal models of all 28 extracted symbolic models. However, this symbolic model covers only 2 of the other 27 symbolic models in the sense of Lemma 1. For p from Figure 3.3 we removed one of the three symbolic models by compaction ending up with two symbolic models, which have incomparable sets of covered graphs as for the symbolic models remaining after compaction for $p_1 \wedge p_2$ from Figure 3.1.

Remark 3 (Runtime Complexity of Compaction) *The compaction procedure terminates because only finitely many symbolic models are contained in $\mathcal{S}_{NT,k}$, which have to be checked pairwise.*

To be able to apply Lemma 1 for compaction we first determine a mono. The problem of finding monos amounts to the subgraph isomorphism problem, which is NP-complete. However, because we have (a) types and (b) small graphs (by construction we generate minimal models by operating only on the graphs from the conditions rather than operating on instance graphs) the required time for finding the monos is not the dominant factor. In fact, checking whether the condition $\exists(i_2, c_2 \wedge \neg \mathit{shift}(i_2, \exists(i_1, c_1)))$ is not satisfiable by a finite graph in Lemma 1 is much more involved (and is in general undecidable as discussed before, and, hence, not every required removal can be determined using AutoGraph*): we apply* AutoGraph *on this condition and abort the construction of the nested tableau once a symbolic model could be extracted (finding a symbolic model implies that the condition is satisfiable by a finite model). However, in cases where no finite models but some infinite model exists* AutoGraph *is not suitable for checking the unsatisfiability.*



(a) Extension Candidate G_1 **(b)** Extension Candidate G_2

Figure 4.1: Two extension candidates that include the graph G_0 from Figure 3.1e with obvious monos $m_1 : G_0 \hookrightarrow G_1$ and $m_2 : G_0 \hookrightarrow G_2$

We believe that it is sufficient for the user in many situations to obtain the set of minimal models: AutoGraph *supports this as well by finding the required monos without checking unsatisfiability by finite graphs: this is sufficient for determining minimal models as given in Figure 3.1d and Figure 3.1e.*

Currently we are unable to prevent overlapping/covering of resulting symbolic models on the fly (for example, by preventing some kinds of symmetries) during the computation without a similar impact on runtime.

Remark 4 (Connection to Construction of Tableau) *Compaction can also be understood as a certain form of application of the lifting rule from Definition 4 because the condition $\exists(i_2, c_2 \wedge \neg \mathit{shift}(i_2, \exists(i_1, c_1)))$ to be checked is similar to the operation performed for lifting, which is based on the shifting as well. However, the lifting rule is applied to a conjunction (the elements in a branch determine a conjunction) whereas the compaction is applied to a disjunction (of two symbolic models possibly describing overlapping sets of models). Consequently, the lifting rule computes the* intersection *of the covered finite graphs whereas compaction computes the* $\subseteq$-relation *of the covered finite graphs.*

4.5 Explorability of $\mathcal{S}$

We believe that the exploration of further graphs satisfying a given property p based on the symbolic models is often desireable. In fact, $\mathit{covered}(\mathcal{S})$ can be explored according to Definition 10 by selecting $\langle C, c\rangle \in \mathcal{S}$, by generating a mono $m : C \hookrightarrow G$ to a new finite candidate graph G, and by deciding $m \models c$. Then, an entire automatic exploration can proceed by selecting the symbolic models $\langle C, c\rangle \in \mathcal{S}$ in a round-robin manner using an enumeration of the monos leaving C in each case. However, the exploration may also be guided interactively restricting the considered symbolic models and monos.

For example, consider p_2 from Figure 3.1c for which the algorithm $\mathcal{A}$ returns a single symbolic model $\langle G_0, c_0\rangle$ of which the minimal model is given in Figure 3.1c. In an interactive exploration we may want to decide whether the two graphs given in Figure 4.1 also satisfy p_2. In fact, because $m_1 \models c_0$ and $m_2 \not\models c_0$ we derive $G_1 \models p_2$ and $G_2 \not\models p_2$ as expected.

Remark 5 (Feasibility of Exploration) *An* entire *enumeration is often not feasible, since many properties (such as true and the other properties considered in this report)* *have infinitely many models. However, we believe that it may prove useful in many application scenarios to obtain a finitely representable guidance to construct every possible finite model if needed. The set of symbolic models represents such a guidance indeed.*

Remark 6 (Further Application Scenarios) *As mentioned above we will take advantage of explorability more explicitly in the future. In particular, it could be adapted to generate large sets of graphs or large, realistic graphs, for example, in the context of performance testing.*

Moreover, in the context of coverage-based testing the minimal models that we derive directly from our symbolic models are not necessarily already realistic enough to the user. He might want to enlarge the models (possibly interactively) and determine whether this enlargement is consistent with the specification. However, we believe that the minimal models of a condition, which we are able to generate, are most likely already reasonable test input sets.

1 : REFUTE-FALSE

$$\frac{\mathit{res} = \exists(m, \wedge(\vee(\emptyset)))}{\emptyset}$$

2 : SELECT-HOOK-FROM-PRE-QUEUE

$$\frac{\mathit{res} = \bot \quad \mathit{q\text{-}pre} = \ell \cdot \ell_s}{\{(\diamond, \ell, \diamond, \ell_s, \diamond, \diamond, \diamond)\}}$$

3 : NO-HOOK

$$\frac{\mathit{res} = \bot \quad \mathit{inp} = \wedge()}{\emptyset}$$

4 : LIFT-NEGATIVE-LITERAL-INTO-BRANCHING-RESULT

$$\frac{\mathit{res} = \exists(m, c) \quad \mathit{neg} = \ell \cdot \ell_s}{\{(\diamond, \exists(m, [c \wedge c']), \ell_s, \diamond, \diamond, \diamond)\}} \; \mathit{shift}(m, \ell) = c'$$

5 : LIFT-POSITIVE-LITERALS-FROM-PRE-QUEUE

$$\frac{\mathit{res} = \exists(m, c) \quad \mathit{q\text{-}pre} = \ell \cdot \ell_s}{\begin{array}{l} \{(\diamond, \diamond, \diamond, \ell_s, \mathit{q\text{-}post} \cdot \exists(m', [c']), \diamond) \mid \exists(m', c') \in L \wedge \neg \mathit{iso}(m')\} \\ \cup \{(\diamond, \exists(m, [c \wedge \exists(m', c')]), \diamond, \ell_s, \diamond, \diamond) \mid \exists(m', c') \in L \wedge \mathit{iso}(m')\} \end{array}} \; \mathit{shift}(m, \ell) = \vee L$$

6 : CREATE-NESTED-TABLEAU

$$\frac{\mathit{res} = \exists(m, c) \quad \mathit{inp} = \wedge()}{\{(c, \bot, \diamond, \mathit{q\text{-}post}, \lambda, m \circ \mathit{cm})\}}$$

7 : EXTEND-USING-FIRST-CLAUSE

$$\frac{\mathit{inp} = \wedge(\mathit{cl}_1, \mathit{cl}_2, \ldots, \mathit{cl}_n)}{\begin{array}{l} \{(\wedge(\mathit{cl}_2, \ldots, \mathit{cl}_n), \diamond, \diamond, \mathit{q\text{-}pre} \cdot \exists(m, c), \diamond, \diamond) \mid \exists(m, c) \in L\} \\ \cup \{(\wedge(\mathit{cl}_2, \ldots, \mathit{cl}_n), \diamond, \mathit{neg} \cdot \neg\exists(m, c), \diamond, \diamond, \diamond) \mid \neg\exists(m, c) \in L\} \end{array}} \; \mathit{cl}_1 = \vee L$$

Figure 4.2: Implemented construction rules: ℓ is a literal, ℓ_s is a sequence of literals, L is a set of literals, cl_i is a clause, and $\diamond$ is the unchanged value from the input

5 Implementation

We implemented the algorithm $\mathcal{A}$ platform-independently using Java as our new tool AutoGraph using xsd-based [39] input/output-format.

While some elementary constructions used (such as computing CNF, existence of monos, and pair factorization) have exponential worst case executing time, we believe, based on our tool-based evaluation, that in many practical applications the runtime will be acceptable. Furthermore, we optimized performance by exploiting parallelizability of the tableaux construction (by considering each nested branch in parallel) and of the compaction of the sets of symbolic models (by considering each pair of symbolic models in parallel).

In Figure 5.1 we have visualized the runtime required by AutoGraph for the condition $p_1 \wedge p_2$ from Figure 3.1. The minimal models derived using AutoGraph for p_1, p_2, and $p_1 \wedge p_2$ from Figure 3.1 are given in Figure 3.1d and Figure 3.1e. We believe that the 24 threads will be appropriate when using bigger examples in the future: for the considered example the runtime is not decreasing further because the width of the tableau is not large enough. For p from Figure 3.3 AutoGraph terminates in negligable time.

To limit memory consumption we discard parts of the nested tableau not required for the subsequent computation, which generates the symbolic models, as follows. The implemented algorithm operates on a queue (used to enforce the breadth-first construction) of configurations where each configuration represents the last branch of a nested branch of the nested tableau currently constructed (the parts of the nested tableau not given by theses branches are thereby not represented in memory). The algorithm starts with a single initial configuration and terminates if the queue of configurations is empty.

A configuration contains the information necessary to continue the further construction of the nested tableau (also ensuring fair selection of hooks) and to extract the symbolic models whenever one is obtained.

A configuration of the implementation is a tuple containing six elements $(\mathit{inp}, \mathit{res}, \mathit{neg}, \mathit{q\text{-}pre}, \mathit{q\text{-}post}, \mathit{cm})$ where *inp* is a condition c over C in CNF and is the remainder of the condition currently constructed (where clauses used already are removed), *res* is $\bot$ or a positive literal $\exists(m : C \hookrightarrow D, c')$ into which the other literals from the branch are lifted, *neg* is a list of negative literals over C from clauses already handled (this list is emptied as soon as a positive literal has been chosen for *res*), *q-pre* is a queue of positive literals over C from which the first element is chosen for the *res* component, *q-post* is a queue of positive literals: once *res* is a chosen positive literal $\exists(m : C \hookrightarrow D, c')$ we shift the elements from *q-pre* over m to obtain elements of *q-post*, and *cm* is the composition of the morphisms from the openers

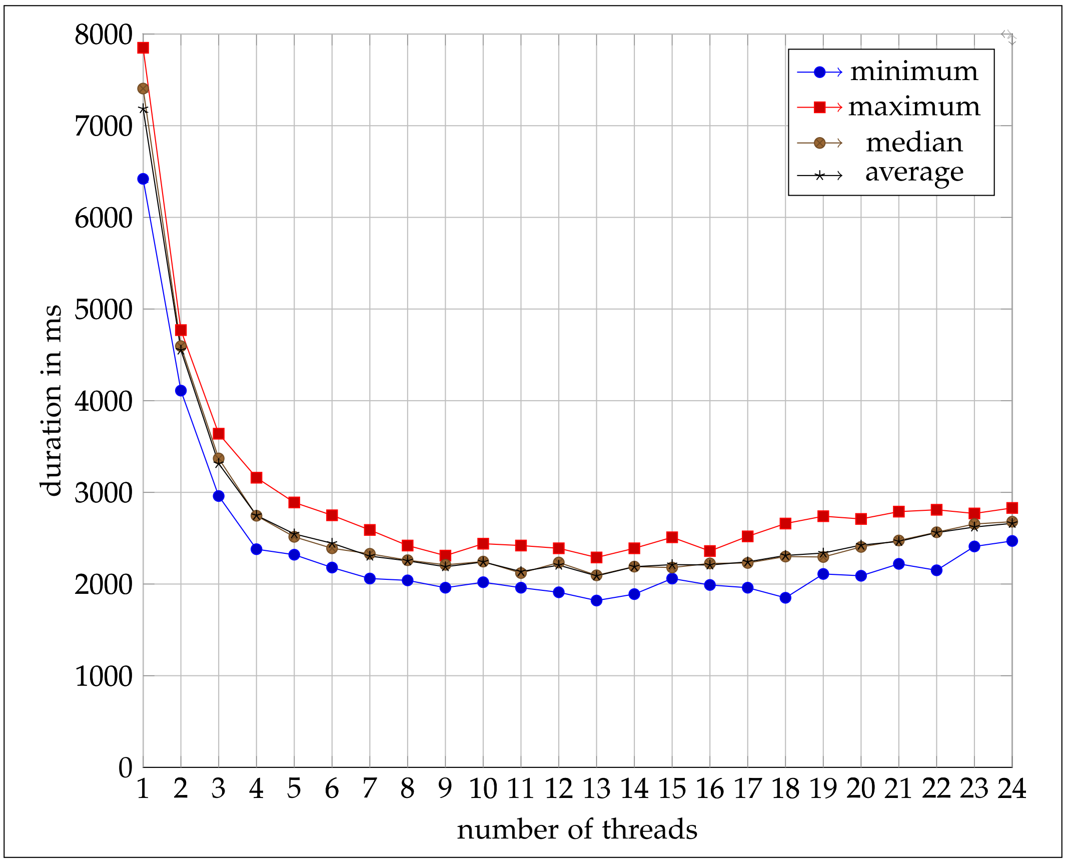


Figure 5.1: Runtime of AutoGraph for the condition $p_1 \wedge p_2$ from Figure 3.1. The following figure provides a complete overview using 1 to 24 threads on our machine and 10 iterations. We used the machine with 256 GB DDR4 and 2 × E5-2643 Xeon @ 3.4 GHz × 6 cores × 2 threads. When also executing the compaction we required about 129 s using 24 threads.

of the nested branch constructed so far and is used to obtain eventually symbolic models (if they exist).

Given a condition c over C the single initial configuration is $(c, \perp, \lambda, \lambda, \lambda, id_C)$. The implemented construction rules operating on these configurations are given in Figure 4.2. Given a configuration c we check the rules in the order given for applicability and apply only the first rule found. For each rule, applicability is determined by the conditions above the line and each rule results in a set of configurations given below the rule.

Rule 1 stops further generation if the current result is unsatisfiable. Rule 2 ensures that hooks are selected from the queue (if the queue is not empty) to ensure fairness of hook selection. Rule 3 if the queue can not be used to select a hook and no clause remains, the nested branch is terminated and a symbolic model can be extracted by taking $\langle codomain(cm), \wedge neg \rangle$. Rule 4 implements the lifting rule (see Definition 4) for negative literals taken from *neg*. Rule 5 implements the lifting rule (see Definition 4) for positive literals taken from *q-pre*; if the morphism of the resulting positive literal is an isomorphism, as forbidden for literals in CNF, we move an equivalent condition in CNF into the current hook (also implementing the lift rule) instead of moving the literal to the queue *q-post*. Rule 6 implements the nesting rule (see Definition 6). Rule 7 deterministically implements the extension rule (see Definition 4) constructing for each literal of the first clause a new configuration to represent the different nested branches.

For soundness reconsider Definition 13 where the set R used in the condition $\wedge R$ recovers the desired information similarly to how it is captured in the configurations. The separation into different elements in the configurations then allows for queue handling and determinization.

6 Conclusion and Outlook

We presented a symbolic model generation procedure for graph properties being equivalent to FOL on graphs. Our algorithm is innovative in the sense that it is designed to generate a finite set of symbolic models that is sound, complete (upon termination), compact, minimally representable, and flexibly explorable. Moreover, the algorithm is highly parallelizable. The approach is implemented in a new tool, called AutoGraph.

As future work we aim at applying, evaluating, and optimizing our approach further w.r.t. different application scenarios from the graph database domain [40] as presented in this paper, but also to other domains such as model-driven engineering, where our approach can be used, for example, to generate test models for model transformations [2, 12, 25]. We also aim at generalizing our approach to more expressive graph properties able to encode, for example, path-related properties [24, 30, 31]. Finally, the work on exploration and compaction of extracted symbolic models as well as reducing their number during tableau construction is an ongoing task.

References

[1] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski. “Clafer: unifying class and feature modeling”. In: *Software and System Modeling* 15.3 (2016), pages 811–845. DOI: 10.1007/s10270-014-0441-1.

[2] B. Baudry. “Testing Model Transformations: A case for Test Generation from Input Domain Models”. In: *Model Driven Engineering for Distributed Real-time Embedded Systems*. Hermes, 2009.

[3] T. Beyhl, D. Blouin, H. Giese, and L. Lambers. “On the Operationalization of Graph Queries with Generalized Discrimination Networks”. In: *Graph Transformation - 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*. Edited by R. Echahed and M. Minas. Volume 9761. Lecture Notes in Computer Science. Springer, 2016, pages 170–186. ISBN: 978-3-319-40529-2. DOI: 10.1007/978-3-319-40530-8_11.

[4] B. Courcelle. “The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic”. In: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. Edited by G. Rozenberg. World Scientific, 1997, pages 313–400. ISBN: 98102288-48.

[5] R. Echahed and M. Minas, editors. *Graph Transformation - 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*. Volume 9761. Lecture Notes in Computer Science. Springer, 2016. ISBN: 978-3-319-40529-2. DOI: 10.1007/978-3-319-40530-8.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer-Verlag, 2006.

[7] H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas. “$\mathcal{M}$-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation”. In: *Mathematical Structures in Computer Science* 24.4 (2014). DOI: 10.1017/S0960129512000357.

[8] H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas. “$\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence”. In: *Fundam. Inform.* 118.1–2 (2012), pages 35–63. DOI: 10.3233/FI-2012-705.

[9] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat-Pérez, M. Pham, and P. A. Boncz. “The LDBC Social Network Benchmark: Interactive Workload”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June*

4, 2015. Edited by T. K. Sellis, S. B. Davidson, and Z. G. Ives. ACM, 2015, pages 619–630. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742786.

[10] H. Giese and B. König, editors. *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*. Volume 8571. Lecture Notes in Computer Science. Springer, 2014. ISBN: 978-3-319-09107-5. DOI: 10.1007/978-3-319-09108-2.

[11] M. Gogolla and F. Hilken. "Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool". In: *Modellierung 2016, 2.-4. März 2016, Karlsruhe*. Edited by A. Oberweis and R. H. Reussner. Volume 254. LNI. GI, 2016, pages 205–220. ISBN: 978-3-88579-648-0.

[12] C. A. González and J. Cabot. "Test Data Generation for Model Transformations Combining Partition and Constraint Analysis". In: *Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings*. Edited by D. D. Ruscio and D. Varró. Volume 8568. Lecture Notes in Computer Science. Springer, 2014, pages 25–41. ISBN: 978-3-319-08788-7. DOI: 10.1007/978-3-319-08789-4_3.

[13] A. Habel, R. Heckel, and G. Taentzer. "Graph Grammars with Negative Application Conditions". In: *Fundam. Inform.* 26.3/4 (1996), pages 287–313. DOI: 10.3233/FI-1996-263404.

[14] A. Habel and K. Pennemann. "Correctness of high-level transformation systems relative to nested conditions". In: *Mathematical Structures in Computer Science* 19.2 (2009), pages 245–296. DOI: 10.1017/S0960129508007202.

[15] R. Hähnle. "Tableaux and Related Methods". In: *Handbook of Automated Reasoning (in 2 volumes)*. Edited by J. A. Robinson and A. Voronkov. Elsevier and MIT Press, 2001, pages 100–178. ISBN: 0-444-50813-9, 0-262-18223-8.

[16] R. Heckel and A. Wagner. "Ensuring consistency of conditional graph rewriting - a constructive approach". In: *Electr. Notes Theor. Comput. Sci.* 2 (1995), pages 118–126. DOI: 10.1016/S1571-0661(05)80188-4.

[17] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2016. ISBN: 9780262528900.

[18] D. Jackson, A. Milicevic, E. Torlak, E. Kang, and J. Near. *alloy: a language & tool for relational models*. 2017. URL: http://alloy.mit.edu/alloy/ (visited on 2017-01-20).

[19] E. K. Jackson, T. Levendovszky, and D. Balasubramanian. "Reasoning about Metamodeling with Formal Specifications and Automatic Proofs". In: *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*. Edited by J. Whittle, T. Clark, and T. Kühne. Volume 6981. Lecture Notes in Computer Science. Springer, 2011, pages 653–667. ISBN: 978-3-642-24484-1. DOI: 10.1007/978-3-642-24485-8_48.

[20] E. K. Jackson and J. Sztipanovits. "Constructive Techniques for Meta- and Model-Level Reasoning". In: *Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings*. Edited by G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil. Volume 4735. Lecture Notes in Computer Science. Springer, 2007, pages 405–419. ISBN: 978-3-540-75208-0. DOI: 10.1007/978-3-540-75209-7_28.

[21] C. Krause, D. Johannsen, R. Deeb, K. Sattler, D. Knacker, and A. Niadzelka. "An SQL-Based Query Language and Engine for Graph Pattern Matching". In: *Graph Transformation - 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*. Edited by R. Echahed and M. Minas. Volume 9761. Lecture Notes in Computer Science. Springer, 2016, pages 153–169. ISBN: 978-3-319-40529-2. DOI: 10.1007/978-3-319-40530-8_10.

[22] S. Lack and P. Sobocinski. "Adhesive and quasiadhesive categories". In: *ITA* 39.3 (2005), pages 511–545. DOI: 10.1051/ita:2005028.

[23] L. Lambers and F. Orejas. "Tableau-Based Reasoning for Graph Properties". In: *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*. Edited by H. Giese and B. König. Volume 8571. Lecture Notes in Computer Science. Springer, 2014, pages 17–32. ISBN: 978-3-319-09107-5. DOI: 10.1007/978-3-319-09108-2_2.

[24] A. Milicevic, J. P. Near, E. Kang, and D. Jackson. "Alloy*: A General-Purpose Higher-Order Relational Constraint Solver". In: *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. Edited by A. Bertolino, G. Canfora, and S. G. Elbaum. IEEE Computer Society, 2015, pages 609–619. ISBN: 978-1-4799-1934-5. DOI: 10.1109/ICSE.2015.77.

[25] A. Mougenot, A. Darrasse, X. Blanc, and M. Soria. "Uniform Random Generation of Huge Metamodel Instances". In: *Model Driven Architecture - Foundations and Applications, 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23-26, 2009. Proceedings*. Edited by R. F. Paige, A. Hartman, and A. Rensink. Volume 5562. Lecture Notes in Computer Science. Springer, 2009, pages 130–145. ISBN: 978-3-642-02673-7. DOI: 10.1007/978-3-642-02674-4_10.

[26] F. Orejas, H. Ehrig, and U. Prange. "Reasoning with graph constraints". In: *Formal Asp. Comput.* 22.3-4 (2010), pages 385–422. DOI: 10.1007/s00165-009-0116-9.

[27] K. Pennemann. "An Algorithm for Approximating the Satisfiability Problem of High-level Conditions". In: volume 213. 1. 2008, pages 75–94. DOI: 10.1016/j.entcs.2008.04.075.

[28] K.-H. Pennemann. *Development of Correct Graph Transformation Systems, PhD Thesis*. Dept. Informatik, Univ. Oldenburg, 2009.

[29] K. Pennemann. “Resolution-Like Theorem Proving for High-Level Conditions”. In: *Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings*. Edited by H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer. Volume 5214. Lecture Notes in Computer Science. Springer, 2008, pages 289–304. ISBN: 978-3-540-87404-1. DOI: 10.1007/978-3-540-87405-8_20.

[30] C. M. Poskitt and D. Plump. “Verifying Monadic Second-Order Properties of Graph Programs”. In: *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*. Edited by H. Giese and B. König. Volume 8571. Lecture Notes in Computer Science. Springer, 2014, pages 33–48. ISBN: 978-3-319-09107-5. DOI: 10.1007/978-3-319-09108-2_3.

[31] H. Radke. “HR* Graph Conditions Between Counting Monadic Second-Order and Second-Order Graph Formulas”. In: *ECEASST* 61 (2013).

[32] H. Radke, T. Arendt, J. S. Becker, A. Habel, and G. Taentzer. “Translating Essential OCL Invariants to Nested Graph Constraints Focusing on Set Operations”. In: *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 21-23, 2015. Proceedings*. Edited by F. Parisi-Presicce and B. Westfechtel. Volume 9151. Lecture Notes in Computer Science. Springer, 2015, pages 155–170. ISBN: 978-3-319-21144-2. DOI: 10.1007/978-3-319-21145-9_10.

[33] A. Rensink. “Representing First-Order Logic Using Graphs”. In: *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28 - October 2, 2004, Proceedings*. Edited by H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg. Volume 3256. Lecture Notes in Computer Science. Springer, 2004, pages 319–335. ISBN: 3-540-23207-9. DOI: 10.1007/978-3-540-30203-2_23.

[34] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997. ISBN: 98102288-48.

[35] R. Salay and M. Chechik. “A Generalized Formal Framework for Partial Modeling”. In: *Fundamental Approaches to Software Engineering - 18th International Conference, FASE 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Edited by A. Egyed and I. Schaefer. Volume 9033. Lecture Notes in Computer Science. Springer, 2015, pages 133–148. ISBN: 978-3-662-46674-2. DOI: 10.1007/978-3-662-46675-9_9.

[36] S. Schneider, L. Lambers, and F. Orejas. “Symbolic Model Generation for Graph Properties (to appear)”. In: *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Edited by M. Huisman and J. Rubin. Lecture Notes in Computer Science. Springer, 2017.

[37] O. Semeráth, A. Vörös, and D. Varró. "Iterative and Incremental Model Generation by Logic Solvers". In: *Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. Edited by P. Stevens and A. Wasowski. Volume 9633. Lecture Notes in Computer Science. Springer, 2016, pages 87–103. ISBN: 978-3-662-49664-0. DOI: `10.1007/978-3-662-49665-7_6`.

[38] The Linked Data Benchmark Council (LDBC). *Social Network Benchmark*. 2016. URL: `http://ldbcouncil.org/benchmarks/snb` (visited on 2017-01-16).

[39] The World Wide Web Consortium (W3C). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. 2012.

[40] P. T. Wood. "Query languages for graph databases". In: *SIGMOD Record* 41.1 (2012), pages 50–60. DOI: `10.1145/2206869.2206879`.

A Example Based Comparison with Alloy

We compare our tool AutoGraph with the tool Alloy [17], following the translation-based approach mentioned in chapter 2, w.r.t. the expressiveness of the input, the computed results, and performance. For this comparison we make use of the two graph properties p_1 and p_2 from Figure 3.1. This comparison demonstrates the key differences and similarities between the two tools.

AutoGraph is a tool introduced in this report and the companion paper [36]. AutoGraph operates on typed graph conditions and generates a set of symbolic models, which is sound, complete (provided termination), minimally representable, compact, and explorable.

Alloy (that is, the Alloy analyzer) is a tool operating on the alloy language, which is an expressive logic based on the notion of relations and is designed for describing and exploring structures [18]. The Alloy analyzer works by reduction to SAT also employing Kodkod as a model finding engine [18].

We follow the translation-based approach and translate the two graph properties as a first step in Alloy as follows.

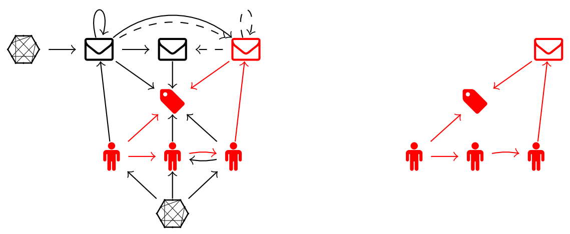
```
// typegraph
sig USER {}
sig TAG {}
sig FORUM {}
sig POST {}
sig knows {src : USER, trg : USER}
sig hasInterest {src : USER, trg : TAG}
sig hasMember {src : FORUM, trg : USER}
sig container {src : FORUM, trg : POST}
sig hasCreator {src : POST, trg : USER}
sig link {src : POST, trg : POST}
sig successor {src : POST, trg : POST}
sig hasTag {src : POST, trg : TAG}

// no parallel edges
pred no_dup1 {all e1 : knows | all e2 : knows |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup2 {all e1 : hasInterest | all e2 : hasInterest |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup3 {all e1 : hasMember | all e2 : hasMember |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup4 {all e1 : container | all e2 : container |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup5 {all e1 : hasCreator | all e2 : hasCreator |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup6 {all e1 : link | all e2 : link |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup7 {all e1 : successor | all e2 : successor |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup8 {all e1 : hasTag | all e2 : hasTag |
    e1 != e2 => e1.src != e2.src || e1.trg != e2.trg}
pred no_dup {
    no_dup1
    && no_dup2
    && no_dup3
    && no_dup4
    && no_dup5
    && no_dup6
    && no_dup7
    && no_dup8
}

// abbreviations
pred true {}
pred false {not true }

// first property
pred prop1 {
    one u1 : USER |
    one t1 : TAG |
        (one u2 : USER |
        one p1 : POST |
        one e1 : knows |
        one e2 : hasCreator |
        one e3 : hasTag |
        u1 != u2
        && e1.src = u1 && e1.trg = u2
        && e2.src = p1 && e2.trg = u2
        && e3.src = p1 && e3.trg = t1
        && ! (one p2 : POST |
            one e4 : hasTag |
            one e5 : link |
            e4 != e3
            && p1 != p2
            && e4.src = p2 && e4.trg = t1
            && e5.src = p1 && e5.trg = p2
            )
        && (one p2 : POST |
            one e4 : successor |
            p1 != p2
            && e4.src = p1 && e4.trg = p2
            )
        )
    && (all u2 : USER |
        all p1 : POST |
        all p2 : POST |
        all e1 : knows |
        all e2 : hasCreator |
        all e3 : hasTag |
        all e4 : successor |
        u1 != u2
        && p1 != p2
        && e1.src = u1 && e1.trg = u2
        && e2.src = p1 && e2.trg = u2
        && e3.src = p1 && e3.trg = t1
        && e4.src = p1 && e4.trg = p2
        => (one p3 : POST |
            one e4 : link |
            one e5 : hasTag |
            e3 != e5
            && p1 != p3
            && p2 != p3
            && e4.src = p1 && e4.trg = p3
            && e5.src = p3 && e5.trg = t1
            )
        )
}

// second property
pred prop2 {
    one u1 : USER |
    one u2 : USER |
    one u3 : USER |
    one e1 : knows |
    one e2 : knows |
    e1 != e2
    && u1 != u2
    && u1 != u3
    && u2 != u3
    && e1.src = u1 && e1.trg = u2
    && e2.src = u2 && e2.trg = u3
    && (
        one t1 : TAG |
        one p1 : POST |
        one e3 : hasCreator |
        one e4 : hasTag |
        one e5 : hasInterest |
        e3.src = p1 && e3.trg = u3
        && e4.src = p1 && e4.trg = t1
        && e5.src = u1 && e5.trg = t1
        )
    && ! (one e3 : knows |
        e1!=e3
        && e2!=e3
        && e3.src = u1 && e3.trg = u3
        )
```

```
127    && (all t1 : TAG |
128         one p1 : POST |
129         one e3 : hasCreator |
130         one e4 : hasTag |
131         one e5 : hasInterest |
132         e3.src = p1 && e3.trg = u3
133         && e4.src = p1 && e4.trg = t1
134         && e5.src = u1 && e5.trg = t1
135         )
136 }
137
138 // command to generate model
139 run {prop1 && prop2 && no_dup} for 3
```

The tool Alloy then returns the *first* model based on the command in line 139 in the previous listing with the following output.

```
Executing "Run run$1 for 3"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   71331 vars. 180 primary vars. 199122 clauses. 2184ms.
   Instance. found. Predicate is consistent. 248ms.
```

In our notation this result model can be visualized as on the left side in the following diagram (the right side corresponds to the minimal model computed by AutoGraph as given in Figure 3.1e and is contained in the left side).



This diagram shows the difference between finding some first model in Alloy (left side) and finding a symbolic, minimally representable model in AutoGraph (right side). Also note that Alloy is able to generate further models by delegating this request to the connected SAT-solver. This technique allows a fast generation of further models. However, due to the scope of 3 given in line 139 Alloy will never generate models with more than 3 elements of a certain type. Hence, to find bigger models during exploration Alloy would have to be restarted for bigger scopes, which then requires time-consuming conversions to CNF again.

To get first information on the runtime of both tools we performed an experiment to compare both regarding the execution time for finding *any* first model. For a fair comparison we temporarily changed AutoGraph to terminate upon finding the first symbolic model and considered 5 runs of both tools. In this experiment Alloy requires 2456 ms (2199 ms for the conversion to CNF and 257 ms for finding the first model) and AutoGraph requires 846 ms. These numbers are hard to compare because Alloy then allows for a rather quick further exploration, which is not yet implemented in AUTOGRAPH, but is confronted with the limiting scope size as discussed before. Moreover, AUTOGRAPH returns not only a minimally representable model but a symbolic model.

We summarize our observations:

- *Input:* Every typed graph condition can be encoded in ALLOY but, on the other hand, we do not expect that every ALLOY-input can be properly rephrased as a typed graph condition. Hence, we conclude that AUTOGRAPH operates on a more restricted setting compared to ALLOY.
- *Results:* AUTOGRAPH is capable of obtaining minimal, symbolic models, which allow for a straightforward exploration of further models whereas ALLOY generates models for a given scope not necessarily determining minimal models. Also, AUTOGRAPH allows for the refutation of a given formula, which is not directly given in ALLOY where nonexistence of models is also bound to scope sizes. Hence, we conclude that AUTOGRAPH computes in this sense stronger results compared to ALLOY.
- *Performance:* Neglecting the different results we observed for our running example comparable runtimes. However, as stated before, AUTOGRAPH already returns stronger results by computing not only a minimally representable model but a symbolic model.

B Proofs

The canonical model can be defined simpler than in [23] if only finite nested branches are considered.

Definition 15 (k-canonical model of nested branch) *Let NT be a nested tableau for condition c over C and NB be a nested branch starting with branches $B_1, \dots, B_k$ of tableaux $T_1, \dots, T_k$ in NT. If C_k is the context of B_k and $\exists(m_2, c_2), \dots, \exists(m_k, c_k)$ are the openers of $T_2, \dots, T_k$, then $\langle C_k, m_k \circ \dots \circ m_2 \circ id_C : C \hookrightarrow C_k \rangle$ is the k-*canonical model *of NB.*

Lemma 2 (reflexivity of sufficient graph condition) *If c is a condition over C, then $c \vdash_{id_C} c$.*

Proof 1 (of Lemma 2) *By Definition 8 we need to show: "for all monos $m_1 : C \hookrightarrow G$ and $m_2 : C \hookrightarrow G$ such that $m_2 \circ id_C = m_1$ it holds that $m_2 \models c$ implies $m_1 \models c$."*
Fix monos $m_1 : C \hookrightarrow G$ and $m_2 : C \hookrightarrow G$ such that $m_2 \circ id_C = m_1$ and $m_2 \models c$.
We need to show: "$m_1 \models c$."
We have: $m_1 = m_2$ from $m_2 \circ id_C = m_1$.
Hence: $m_1 \models c$ from $m_2 \models c$.

Lemma 3 (weakening of sufficient graph condition) *If c and c' are conditions over C and for every mono $m : C \hookrightarrow G$ it holds that $m \models c' \rightarrow c$, then $c' \vdash_{id_C} c$.*

Proof 2 (of Lemma 3) *By Definition 8 we need to show: "for all monos $m_1 : C \hookrightarrow G$ and $m_2 : C \hookrightarrow G$ such that $m_2 \circ id_C = m_1$ it holds that $m_2 \models c'$ implies $m_1 \models c$."*
Fix monos $m_1 : C \hookrightarrow G$ and $m_2 : C \hookrightarrow G$ such that $m_2 \circ id_C = m_1$ and $m_2 \models c'$.
We need to show: "$m_1 \models c$."
We have: $m_1 = m_2$ from $m_2 \circ id_C = m_1$.
We have: $m_1 \models c'$ from $m_2 \models c'$.
We have: $m_1 \models c' \rightarrow c$ from the assumption.
We have: $m_1 \models c'$ implies $m_1 \models c$ by Definition 2.
Hence: $m_1 \models c$ from $m_1 \models c'$ and ($m_1 \models c'$ implies $m_1 \models c$).

Lemma 4 (soundness of shift) *If c is a condition over C and $m : C \hookrightarrow D$ is a mono, then $shift(m, c) \vdash_m c$.*

Proof 3 (of Lemma 4) *By Definition 8 we need to show: "$m : C \hookrightarrow D$ is a mono, and for all monos $m_1 : C \hookrightarrow G$ and $m_2 : D \hookrightarrow G$ such that $m_2 \circ m = m_1$ it holds that $m_2 \models shift(m, c)$ implies $m_1 \models c$."*
This follows from: "$m : C \hookrightarrow D$ is a mono, and for all monos $m_1 : C \hookrightarrow G$ and $m_2 : D \hookrightarrow G$ such that $m_2 \circ m = m_1$ it holds that $m_2 \models shift(m, c)$ is equivalent to $m_1 \models c$."

This follows from: "$m : C \hookrightarrow D$ is a mono, and for every mono $m_2 : D \hookrightarrow G$ it holds that $m_2 \models shift(m, c)$ is equivalent to $m_2 \circ m \models c$."

This follows from: [23, Lemma 3].

Lemma 5 (soundness of lifting) *If c_1 and $\exists(m : C \hookrightarrow D, c_2)$ are conditions over C, then $\exists(m, c_2 \land shift(m, c_1)) \vdash_{id_C} c_1 \land \exists(m, c_2)$.*

Proof 4 (of Lemma 5) *By application of Lemma 3 it is sufficient to show that for every mono $p : C \hookrightarrow G$ it holds that $p \models \exists(m, c_2 \land shift(m, c_1)) \to c_1 \land \exists(m, c_2)$.*

Fix a mono $p : C \hookrightarrow G$.

- *case $p \models c_1$:*
  - *case $p \models \exists(m, c_2)$: the implication holds because the conclusion holds.*
  - *case $p \not\models \exists(m, c_2)$:*
    we need to show $p \models \neg\exists(m, c_2 \land shift(m, c_1))$, which follows from $p \not\models \exists(m, c_2 \land shift(m, c_1))$. Assume for the contradiction that $p \models \exists(m, c_2 \land shift(m, c_1))$. That is, there must be a $q' : D \hookrightarrow G$ such that $q' \models c_2 \land shift(m, c_1)$ and $q' \circ m = p$. Hence, $q' \models c_2$. Hence, $p \models \exists(m, c_2)$ by using the same q' establishing the contradiction.
- *case $p \not\models c_1$:*
  we need to show $p \models \neg\exists(m, c_2 \land shift(m, c_1))$, which follows from $p \not\models \exists(m, c_2 \land shift(m, c_1))$. Assume for the contradiction that $p \models \exists(m, c_2 \land shift(m, c_1))$. That is, there must be a $q' : D \hookrightarrow G$ such that $q' \models c_2 \land shift(m, c_1)$ and $q' \circ m = p$. Hence, $q' \models shift(m, c_1)$. By application of Lemma 4 we get: $shift(m, c_1) \vdash_m c_1$. Hence, for every monos $m_C : C \hookrightarrow G$ and $m_D : D \hookrightarrow G$ holds $m_D \circ m = m_C$ implies $m_D \models shift(m, c_1)$ implies $m_C \models c_1$. Hence, using p for m_C and q' for m_D: $q' \circ m = p$ implies $q' \models shift(m, c_1)$ implies $p \models c_1$. This establishes the contradiction.

Lemma 6 (transitivity of sufficient graph condition) *If c_A is a condition over A, c_B is a condition over B, c_C is a condition over C, $m_1 : A \hookrightarrow B$ is a mono, $m_2 : B \hookrightarrow C$ is a mono, $c_B \vdash_{m_1} c_A$, and $c_C \vdash_{m_2} c_B$, then $c_C \vdash_{m_2 \circ m_1} c_A$.*

Proof 5 (of Lemma 6) *From $c_B \vdash_{m_1} c_A$ we know that: for every monos $m_B : B \hookrightarrow G$ and $m_A : A \hookrightarrow G$ holds $m_B \circ m_1 = m_A$ implies $m_B \models c_B$ implies $m_A \models c_A$.*

From $c_C \vdash_{m_2} c_B$ we know that: for every monos $m_C : C \hookrightarrow G$ and $m_B : B \hookrightarrow G$ holds $m_C \circ m_2 = m_B$ implies $m_C \models c_C$ implies $m_B \models c_B$.

For $c_C \vdash_{m_2 \circ m_1} c_A$ we need to show that: for every monos $m_C : C \hookrightarrow G$ and $m_A : A \hookrightarrow G$ holds $m_C \circ (m_2 \circ m_1) = m_A$ implies $m_C \models c_C$ implies $m_A \models c_A$.

Fix monos $m_C : C \hookrightarrow G$ and $m_A : A \hookrightarrow G$ such that $m_C \circ (m_2 \circ m_1) = m_A$ and $m_C \models c_C$.

We show that $m_A \models c_A$. We apply the first assumption by using m_A for m_A and $m_C \circ m_2$ for m_B: we get $(m_C \circ m_2) \circ m_1 = m_A$ implies $(m_C \circ m_2) \models c_B$ implies $m_A \models c_A$. Here, the current goal $m_A \models c_A$ appears as conclusion.

We show that $(m_C \circ m_2) \circ m_1 = m_A$ and $(m_C \circ m_2) \models c_B$. Firstly, $(m_C \circ m_2) \circ m_1 = m_A$ holds as stated before. Secondly, $(m_C \circ m_2) \models c_B$ holds as follows.

We apply the second assumption by using m_C for m_C and $m_C \circ m_2$ for m_B: we get $m_C \circ m_2 = m_C \circ m_2$ implies $m_C \models c_C$ implies $(m_C \circ m_2) \models c_B$. Here, the current goal $(m_C \circ m_2) \models c_B$ appears as conclusion.

We show that $m_C \circ m_2 = m_C \circ m_2$ and $m_C \models c_C$. Firstly, $m_C \circ m_2 = m_C \circ m_2$ is trivial. Secondly, $m_C \models c_C$ holds as stated before.

Lemma 7 (satisfaction to sufficient graph condition) *If $\exists(m : C \hookrightarrow A, c)$ is a condition over C then $c \vdash_m \exists(m, c)$.*

Proof 6 (of Lemma 7) *Fix monos $m_A : A \hookrightarrow G$ and $m_C : C \hookrightarrow G$ such that $m_A \circ m = m_C$ and $m_A \models c$. We need to show $m_C \models \exists(m, c)$. This holds trivially.*

To obtain soundness of extracted symbolic models we need a soundness result for nested tableau that are "prefixes of semi-saturated tableau".

Lemma 8 (k-soundness of nested tableau) *If NT is a nested tableau for a condition c over C, NB is a $k-1$-semi-saturated nested branch of length $n \leq k$, NB has branch B of length at least k' at index n, $\langle C, c' \rangle$ is the k'-remainder of B, $\langle G_k, q_k : C \hookrightarrow G_k \rangle$ is the k-canonical model of NB, then $c' \vdash_{q_k} c$.*

Proof 7 (of Lemma 8) *Let $\langle C, c' \rangle = \langle C, \wedge R_k^{k'} \rangle$ be the k'-remainder of B. By induction on k.*

- *case $k = 0$: the canonical model G_0 equals C and $q_0 = id_C$ by definition. We prove the property by induction on k'.*
  - *case $k' = 0$: If no tableau rule has been applied R_0^0 contains all clauses of c. Hence, $\wedge(R_0^0) \vdash_{id_C} c$ is equivalent to $c \vdash_{id_C} c$. This holds according to Lemma 2.*
  - *case $k' \to k' + 1$: Let $\langle C, \wedge R_0^{k'} \rangle$ be the k'-remainder of B_0. Let $\langle C, \wedge R_0^{k'+1} \rangle$ be the $k' + 1$-remainder of B_0. An additional node needs to be considered, which is added according to one of the tableau rules. We show $\wedge R_0^{k'+1} \vdash_{id_C} c$. For this we apply Lemma 3 and have to show that for every mono $m : C \hookrightarrow G$ it holds that $m \models \wedge(R_0^{k'+1}) \to \wedge(R_0^{k'})$.*
    - *application of the initialization rule: In this case $R_0^{k'+1} = R_0^{k'} \cup \{true\}$ and, hence, the implication is trivially a tautology.*
    - *application of the extension rule: Assume that from clause cl the literal l has been added. In this case $R_0^{k'+1} = R_0^{k'} - \{cl\} \cup \{l\}$ and, hence, the implication is trivially a tautology.*
    - *application of the lift rule: Assume that the literal l has been lifted into the positive literal $\exists(m, c)$. In this case $R_0^{k'+1} = R_0^{k'} - \{l, \exists(m, c)\} \cup \{\exists(m, [\wedge(c, [shift(m, l)])])\}$ and, hence, the implication is a tautology by Lemma 5.*

- *case $k \to k+1$: Let B_k be the semi-saturated branch at index k in NT. Let $q_k : C \to G_k$ with G_k and $q_{k+1} : C \to G_{k+1}$ and G_{k+1} are the canonical models up to indexes k and $k+1$, respectively. Let R_k be the condition that is collected from B_k according to Definition 13. Obviously, as B_k is semi-saturated the opener $\exists(a_k, c_{k+1})$ taken from B_k (that is, the leaf of B_k) contains all the information in the sense that $\wedge R_k$ is equivalent to $\exists(a_k, c_{k+1})$.*

  By construction of the canonical model $q_{k+1} = a_k \circ q_k$.

  As an assumption we now have that $\wedge R_k \vdash_{q_k} c$.

  We prove the property by induction on k'. However, before considering the two cases of the induction we can simplify the goal such that only the reasoning within the tableau of B_{k+1} is to be handled.

  Let $B^{k'}_{k+1}$ be the prefix of length k' of the branch at index $k+1$. Let $R^{k'}_{k+1}$ be the condition that is collected from $B^{k'}_{k+1}$ according to Definition 13.

  We need to show that $\wedge(R^{k'}_{k+1}) \vdash_{q_{k+1}} c$.

  This follows from: $\wedge(R^{k'}_{k+1}) \vdash_{a_k \circ q_k} c$ since $q_{k+1} = a_k \circ q_k$.

  This follows from: $\wedge(R^{k'}_{k+1}) \vdash_{a_k} \exists(a_k, c_{k+1})$ and $\exists(a_k, c_{k+1}) \vdash_{q_k} c$ according to Lemma 6.

  The second part ($\exists(a_k, c_{k+1}) \vdash_{q_k} c$) holds from $\exists(a_k, c_{k+1}) \equiv \wedge(R_k)$ and $\wedge R_k \vdash_{q_k} c$.

  We need to show the first part: $\wedge(R^{k'}_{k+1}) \vdash_{a_k} \exists(a_k, c_{k+1})$.

  This follows from: $\wedge(R^{k'}_{k+1}) \vdash_{id_{G_{k+1}}} c_{k+1}$ and $c_{k+1} \vdash_{a_k} \exists(a_k, c_{k+1})$ according to Lemma 6.

  The second part ($c_{k+1} \vdash_{a_k} \exists(a_k, c_{k+1})$) holds from Lemma 7.

  We need to show the first part: $\wedge(R^{k'}_{k+1}) \vdash_{id_{G_{k+1}}} c_{k+1}$.

  We now cover the two cases of the induction on k': these two cases are similar to the induction proof on k' for the case of $k = 0$ in the outer induction.

  - *case $k' = 0$: see the "case $k' = 0$:" before replacing R^0_0 by R^0_{k+1}, C by G_{k+1}, and c by c_{k+1}.*
  - *case $k' \to k'+1$: see the "case $k' \to k'+1$:" before replacing $B^{k'}_0$ by $B^{k'}_{k+1}$, $R^{k'}_0$ by $R^{k'}_{k+1}$, $R^{k'+1}_0$ by $R^{k'+1}_{k+1}$, C by G_{k+1}, and c by c_{k+1}.*

Proof 8 (of Theorem 1) *According to Definition 11 we need to show covered($\mathcal{S}_{NT,k}$) $\subseteq \{G \mid G \models c \wedge G$ is finite$\}$. According to Definition 10 we need to show $\{G \mid G$ is finite $\wedge \exists\langle C', c'\rangle \in \mathcal{S}_{NT,k}, m : C' \hookrightarrow G \mid m \models c'\} \subseteq \{G \mid G \models c \wedge G$ is finite$\}$. Fix a finite graph G, a symbolic model $\langle C', c'\rangle \in \mathcal{S}_{NT,k}$, and a mono $m : C' \hookrightarrow G$ such that $m \models c'$. We need to show that $G \models c$. Let $i : \emptyset \hookrightarrow G$ be the unique mono. We need to show that $i \models c$. According to Definition 14 we know that there is a k-terminated and k-semi-saturated nested branch NB of NT of length $n \leq k$ such that there is a branch B at index n of length k' in NB such that B is open and B contains no positive literals and $\langle C', c'\rangle$ is the k'-*remainder *of B. Let $\langle G_k, q_k : C \hookrightarrow G_k\rangle$ be the k-canonical model of NB. According to Lemma 8 we have $c' \vdash_{q_k} c$. By application of Definition 8 we know that "If c and c'* *are conditions over $\emptyset$ and C', respectively, $q_k : \emptyset \hookrightarrow C'$ is a mono, and for all monos $i : \emptyset \hookrightarrow G$ and $m : C' \hookrightarrow G$ such that $m \circ q_k = i$ it holds that $m \models c'$ implies $i \models c$." Now, $m \circ q_k = i$ holds by uniqueness of initiality of $\emptyset$ and $m \models c'$ holds as stated above, hence, $i \models c$, which had to be shown.*

Proof 9 (of Theorem 2) Intuitively, we can construct for condition that is satisfied by some finite graph G a corresponding nested branch. If the nested branch would not be finite, the size of the graphs in the hooks would be eventually bigger than the finite graph G (because nonisomophic monomorphisms increase the size with each new tableau). Hence, the corresponding nested branch is finite. Then, from the last branch we can extract a symbolic model covering G.

According to Definition 11 we show covered$(\mathcal{S}_{NT,k}) \supseteq \{G \mid G \models c \wedge G$ is finite$\}$.

According to Definition 10 we need to show $\{G \mid G$ is finite $\wedge\ \exists \langle C', c' \rangle \in \mathcal{S}_{NT,k}, m : C' \hookrightarrow G \mid m \models c'\} \supseteq \{G \mid G \models c \wedge G$ is finite$\}$.

Let G be a finite graph such that $G \models c$. Let $i : \emptyset \hookrightarrow G$ be the unique inclusion. Then, $G \models c$ implies $i \models c$.

We need to show that there are $\langle C', c' \rangle \in \mathcal{S}_{NT,k}$ and $m : C' \hookrightarrow G$ such that $m \models c'$.

*According to Definition 14 we need to show that there is a k-terminated and k-semi-saturated nested branch NB of NT of length $n \leq k$ such that there is a branch B at index n of length k' in NB such that B is open and B contains no positive literals such that $\langle C', c' \rangle$ is the k'-*remainder *of B and that there is $m : C' \hookrightarrow G$ such that $m \models c'$.*

Since $i \models c$ there is a nested branch NB proving this as follows. For the first tableau T apply the initialization rule. Then, apply the extension rule such that there is a branch B containing from each clause some literal satisfied by i.

- *If the condition c is false, then $i \not\models c$ (which is an obvious contradiction to $i \models c$).*
- *If the condition c is true, then $\langle \emptyset, \wedge() \rangle \in \mathcal{S}_{NT,k}$ is a symbolic model and the k'-*remainder *of B. Hence, this nested branch consisting only of B is a 1-terminated and 1-semi-saturated nested branch of NT of length 1 and B is the branch at index 1 of length 1 in NB and B is open and B contains no positive literals and $\langle \emptyset, \wedge() \rangle$ is the 1-*remainder *of B. Moreover, when using $m = id_\emptyset : \emptyset \hookrightarrow G$, then $m \models \wedge()$ trivially. Thus, we have established an appropriate nested branch.*
- *In all other cases (c is not true and not false) there is a positive literal and we select one of these literals as a hook and apply the lift rule with all other literals in B. The resulting opener (let say, $\exists(p_1, c_1)$) is contained as a leaf in the resulting branch B. By construction (tableau rules, lift rule in particular, are sound; confer the proof of [23, lemma 8] and [29, lemma 1]) $i \models \exists(p_1 : \emptyset \hookrightarrow D, c_1)$. Thus, there is $q_1 : D \hookrightarrow G$ such that $q_1 \circ p_1 = i$ and $q_1 \models c_1$. Then, we open a new tableau for c_1 using the opener $\exists(p_1, c_1)$.*

  This construction principle can be applied iteratively often to obtain with each new tableau strictly bigger graphs because the morphisms in openers are not isomorphisms. However, the graphs constructed (domain of q_k) are smaller than the finite graph G. Hence, this construction principle eventually necessarily terminates and we end up in one of the two cases before (in fact, it must be the second case because $i \models c$).

Proof 10 (of Theorem 3) *Consider a nested tableau NT for the graph property c.*

Minimality: *According to Definition 10 for every graph G in the set of covered graphs covered($\langle C', c' \rangle$) there is a mono $m : C' \hookrightarrow G$ satisfying c'.*

Representability: *We need to show that the graph of the symbolic model satisfies the property.*

According to Definition 11 we need to show that for every symbolic model $\langle C', c' \rangle$ contained in $\mathcal{S}_{NT,k}$ it holds that $C' \models c$.

*According to Definition 14 we have for $\langle C', c' \rangle \in \mathcal{S}_{NT,k}$ that there is a k-terminated and k-semi-saturated nested branch NB of NT of length $n \leq k$ such that there is a branch B at index n of length k' in NB such that B is open and B contains no positive literals such that $\langle C', c' \rangle$ is the k'-*remainder *of B.*

According to Definition 13 $\langle C', c' \rangle$ satisfies that $[c']$ contains no clause containing a positive literal, because the k'-remainder contains in this case the root node true and only negative literals because neither positive literals nor false are contained in B.

Let $\langle G_k, q_k : \emptyset \hookrightarrow G_k \rangle$ be the k-canonical model of NB. According to Lemma 8 we have $c' \vdash_{q_k} c$. By application of Definition 8 we know that "If c and c′ are conditions over $\emptyset$ and C', respectively, $q_k : \emptyset \hookrightarrow C'$ is a mono, and for all monos $i : \emptyset \hookrightarrow C'$ and $id_{C'} : C' \hookrightarrow C'$ such that $id_{C'} \circ q_k = i$ it holds that $id_{C'} \models c'$ implies $i \models c$." Now, $id_{C'} \circ q_k = i$ holds by uniqueness of initiality of $\emptyset$ and $id_{C'} \models c'$ holds (because if $\neg(\exists(m : C' \hookrightarrow C'', c''))$ is a negative literal in a clause of $[c']$, then m is no isomorphism and, hence, the triangle to be closed for $id_{C'} \models \exists(m : C' \hookrightarrow C'', c'')$ can not be closed using a mono such that the triangle commutes), hence, $i \models c$ and this implies $C' \models c$ which had to be shown.

Proof 11 (of Lemma 1)

$$\begin{aligned}
&\mathit{covered}(\langle A_1, c_1 \rangle) \supseteq \mathit{covered}(\langle A_2, c_2 \rangle)\\
\Leftarrow&\forall G.\ G \in \mathit{covered}(\langle A_2, c_2 \rangle) \to G \in \mathit{covered}(\langle A_1, c_1 \rangle)\\
\Leftarrow&\nexists G.\ G \in \mathit{covered}(\langle A_2, c_2 \rangle) \wedge G \notin \mathit{covered}(\langle A_1, c_1 \rangle)\\
\Leftarrow&\nexists G.\ G \in \{G \mid G \models \exists(i_2, c_2) \wedge G \textit{ is finite}\} \wedge G \notin \{G \mid G \models \exists(i_1, c_1) \wedge G \textit{ is finite}\}\\
\Leftarrow&\nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge \neg(G \models \exists(i_1, c_1) \wedge G \textit{ is finite})\\
\Leftarrow&\nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge (G \models \neg\exists(i_1, c_1) \vee G \textit{ is not finite})\\
\Leftarrow&\nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge G \models \neg\exists(i_1, c_1)\\
\Leftarrow&\nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2) \wedge \neg\exists(i_1, c_1)\\
\Leftarrow&\nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2 \wedge \mathit{shift}(i_2, \neg\exists(i_1, c_1)))\\
\Leftarrow&\nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2 \wedge \neg\mathit{shift}(i_2, \exists(i_1, c_1)))
\end{aligned}$$

Proof 12 (of Theorem 4) *Firstly, merely removing symbolic models ensures $\mathcal{S} \subseteq \mathcal{S}_{NT,k}$.*

Secondly, the unsatisfiability of the condition $\exists(i_2, c_2 \wedge \neg\mathit{shift}(i_2, \exists(i_1, c_1)))$ by a finite graph is actually also necessary.

$$\begin{aligned}
&\mathit{covered}(\langle A_1, c_1 \rangle) \supseteq \mathit{covered}(\langle A_2, c_2 \rangle)\\
\Rightarrow&\forall G.\ G \in \mathit{covered}(\langle A_2, c_2 \rangle) \to G \in \mathit{covered}(\langle A_1, c_1 \rangle)\\
\Rightarrow&\nexists G.\ G \in \mathit{covered}(\langle A_2, c_2 \rangle) \wedge G \notin \mathit{covered}(\langle A_1, c_1 \rangle)\\
\Rightarrow&\nexists G.\ G \in \{G \mid G \models \exists(i_2, c_2) \wedge G \textit{ is finite}\} \wedge G \notin \{G \mid G \models \exists(i_1, c_1) \wedge G \textit{ is finite}\}
\end{aligned}$$

$$\Rightarrow \nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge \neg(G \models \exists(i_1, c_1) \wedge G \textit{ is finite})$$
$$\Rightarrow \nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge (G \models \neg\exists(i_1, c_1) \vee G \textit{ is not finite})$$
$$\Rightarrow \nexists G.\ G \models \exists(i_2, c_2) \wedge G \textit{ is finite} \wedge G \models \neg\exists(i_1, c_1)$$
$$\Rightarrow \nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2) \wedge \neg\exists(i_1, c_1)$$
$$\Rightarrow \nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2 \wedge \textit{shift}(i_2, \neg\exists(i_1, c_1)))$$
$$\Rightarrow \nexists G.\ G \textit{ is finite} \wedge G \models \exists(i_2, c_2 \wedge \neg\textit{shift}(i_2, \exists(i_1, c_1)))$$

Hence, the fixed-point of iterated application of Lemma 1 ensures compaction directly.

ISBN 978-3-86956-396-1
ISSN 1613-5652