Tableau-Based Reasoning for Graph Properties

Leen Lambers[1] and Fernando Orejas[2]

[1] Hasso Plattner Institut, University of Potsdam, Germany
[2] Dpto de L.S.I., Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. 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. A simple way is based on defining an appropriate encoding of graphs in terms of classical logic. This approach has been followed by Courcelle. The alternative is the definition of a specialized logic, as done by Habel and Pennemann, who defined a logic of nested graph conditions, where graph properties are formulated explicitly making use of graphs and graph morphisms, and which has the expressive power of Courcelle's first order logic of graphs. In particular, in his thesis, Pennemann defined and implemented a sound proof system for reasoning in this logic. Moreover, he showed that his tools outperform some standard provers when working over encoded graph conditions.

Unfortunately, Pennemann did not prove the completeness of his proof system. In this sense, one of the main contributions of this paper is the solution to this open problem. In particular, we prove the (refutational) completeness of a tableau method based on Pennemann's rules that provides a specific theorem-proving procedure for this logic. This procedure can be considered our second contribution. Finally, our tableaux are not standard, but we had to define a new notion of nested tableaux that could be useful for other formalisms where formulas have a hierarchical structure like nested graph conditions.

Keywords: Graph properties, Graph Logic, Automated deduction, Visual modelling, Graph transformation.

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, model-driven engineering where we may need to express properties of graphical models, or the verification of systems whose states are modelled as graphs, or if we need to express properties about sets of semi-structured documents, especially if they are related by links, or the area of graph databases, to express integrity constraints. We can follow two different ways. The first one is based on defining an appropriate encoding of graphs in terms of some existing logic. The second way is based on directly defining a logic where graphs are first-class citizens. An example of the first approach is the work of Courcelle who, in a series of papers (see, e.g. [3]) studied systematically a graph logic defined in terms of first-order (or monadic second-order) logic. In particular, in that approach, graphs are defined axiomatically by means of 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

H. Giese and B. König (Eds.): ICGT 2014, LNCS 8571, pp. 17–32, 2014.

n_2. Graphs can also be defined as terms over a given algebra A as done in the context of Maude [2,1]. The most prominent example of the second approach is the *logic of nested graph conditions*, that we study in this paper, defined by Pennemann and Habel [6], where graph properties are formulated explicitly making use of graphs and graph morphisms. The origins of this approach can be found in the notion of graph constraint [8], introduced in the area of graph transformation, in connection with the notion of (negative) application conditions, as a form to limit the applicability of transformation rules. However, graph constraints have a very limited expressive power, while nested conditions have the same expressive power as Courcelle's first-order graph logic [6]. A similar approach was first introduced by Rensink in [15].

An advantage of encoding graph properties in terms of some existing logic is that we can reason about them by using methods and tools provided by the given logic, while, in the other case, we would need to define specific dedicated methods. In this sense, in [12], we defined a sound and complete proof system for the restrictive case of graph constraints, including the case where the constraints included conditions on graph attributes [11]. Almost simultaneously, in [13,14], Pennemann defined and implemented a sound proof system for reasoning with nested conditions. Unfortunately, in this work the completeness of his approach was not proven. The problem is related with the difficulty to use induction for building a counter-model in the completeness proof. In this sense, one of the main contributions of this paper is the solution to this open problem, since we prove the completeness of a subset of the rules defined by Pennemann. In particular, we prove the (refutational) completeness of a tableau method based on this subset of rules providing a specific theorem-proving procedure for this logic. This procedure can be considered our second contribution. Moreover, our tableaux are not standard, but we had to define a new notion of *nested tableaux*, that solves the difficulties with induction, and that could be useful for other formalisms where formulas have a hierarchical structure like nested graph conditions.

One may question in which sense this kind of work is relevant or interesting, if (apparently) the encoding approach gives the same expressive power in a simpler way. There are two main reasons: generality and efficiency. On the one hand, the logic of nested graph conditions and our results are not restricted to a given kind of graphs. Our graphs can be directed or undirected, typed or untyped, attributed or without attributes, etc. The only condition is that the given category of graphs should be $\mathcal{M}$-adhesive [9,5] satisfying some additional categorical properties that are used in this paper. Actually, this approach applies also to categories of structures that are not graphs, like sets. On the contrary, when using encodings, each kind of graph structure needs a different encoding. On the other hand, a dedicated theorem-prover may be much more efficient than a generic one when used over the given graph encoding. In particular, in [13,14], Pennemann compares the implementation for his proof system with some standard provers, like VAMPIRE, DARWIN and PROVER9, working over encoded graph conditions. The result is that his implementation outperforms the coding approach. Actually, in a considerable amount of examples, the above standard provers were unable to terminate in the same time needed by Pennemann's proof system.

The paper is organized as follows: In Section 2 we present the kind of graph properties that we consider together with some preliminariy results that we need in the rest

of the paper. In Section 3 we then present our new tableau-based reasoning method for graph properties and subsequently in Section 4 we present soundness and completeness for this reasoning method. We conclude the paper with Section 5. Due to space limitations, proofs are only sketched. Detailed proofs can be found in [10].

2 Preliminaries

In this section we present the basic notions for this paper. First we reintroduce the kind of graph properties that we consider. Secondly we introduce a (weak) conjunctive normal form for them together with some shifting results.

For simplicity, we will present all our notions and results in terms of plain directed graphs and graph morphisms[1], i.e.: A *graph* $G = (G^V, G^E, s^G, t^G)$ consists of a set G^V of nodes, a set G^E of edges, a source function $s^G : G^E \to G^V$, and a target function $t^G : G^E \to G^V$. Given the graphs $G = (G^V, G^E, s^G, t^G)$ and $H = (H^V, H^E, s^H, t^H)$, a *graph morphism* $f : G \to H$ is a pair of mappings, $f^V : G^V \to H^V, f^E : G^E \to H^E$ such that $f^V \circ s^G = s^H \circ f^E$ and $f^V \circ t^G = t^H \circ f^E$. A graph morphism $f : G \to H$ is a *monomorphism* if f^V and f^E are injective mappings. Finally, two graph morphisms $m : H \to G$ and $m' : H' \to G$ are *jointly surjective* if $m^V(H^V) \cup m'^V(H'^V) = G^V$ and $m^E(H^E) \cup m'^E(H'^E) = G^E$. Note that all along the paper, unless it is explicitly said, if we say that f is a morphism, we will implicitly assume that f is a monomorphism.

2.1 Graph Properties Expressed in GL

The underlying idea of the graph logic GL, handled in this paper, is that graph properties can be described stating that certain patterns, consisting of graphs and morphisms (actually, inclusions), must be present (or must not be present) in a given graph. For instance, the simplest kind of graph property, $\exists C$, specifies that a given graph G should include (a copy of) C. For instance, the property $\exists C_1$ with C_1 as depicted on the left of Fig. 1 states that a graph should include a node with a loop. More precisely, the fact that a graph G satisfies a graph property α is expressed in term of the existence (or non-existence) of monomorphisms from the graphs included in α to G, such that some conditions are satisfied. For example, a graph G satisfies the former property if there exists a monomorphism $f : C_1 \to G$. Obviously, graph properties can be combined using the standard connectives $\vee$, $\wedge$, and $\neg$. For example, $\neg\exists C_1$, where C_1 is the graph on the left of Fig. 1, specifies that a graph should not include loops. We can also describe more complex properties by using more complex patterns or diagrams. For instance, we may consider that the property depicted in Fig. 1, where h_1 and h_2 are two inclusions between the graphs involved, states that there must exist a node with a loop, such that

[1] In some examples however, for motivation, we deal with typed attributed graphs. Anyhow, following the approach used in [4], it is straightforward to show that our results generalize to a large class of (graphical) structures. The only condition is that the given category of graphs should be $\mathcal{M}$-adhesive [9,5] satisfying the additional property of having a unique $\mathcal{E}' - \mathcal{M}$ pair factorization [5] (needed for Lemma 3) as well as infinite colimits (see Proposition 1 needed for the Completeness Theorem).

for all pairs of edges connected to that node, there exists another pair of edges completing a rectangle. More precisely, a graph G would satisfy this condition if there exists a monomorphism $f : C_1 \to G$ such that for all $f' : C_2 \to G$, with $f = f' \circ h_1$, there exists $f'' : C_2 \to G$, such that $f' = f'' \circ h_2$.

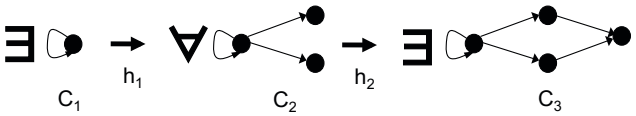


Fig. 1. A graph property

For our convenience, we will express these properties using a nested notation [6] and avoiding the use of universal quantifiers. Moreover, the conditions that we define below are slightly more general than what may seem to be needed. Instead of defining properties about graphs, *nested conditions* define properties of graph monomorphisms.

Definition 1 (condition, nesting level). *Given a finite graph C, a condition over C is defined inductively as follows:*

- *$true$ is a condition over C. We say that $true$ has nesting level* 0.
- *For every monomorphism $a : C \to D$ and condition c_D over a finite graph D with nesting level n such that $n \geq 0$, $\exists(a, c_D)$ is a condition over C with nesting level $n+1$.*
- *Given conditions over C, c_C and c'_C, with nesting level n and n', respectively, $\neg c_C$ and $c_C \wedge c'_C$ are conditions over C with nesting level n and $max(n, n')$, respectively. We restrict ourselves to finite conditions, i.e. each conjunction of conditions is finite.*

We define when a monomorphism $q : C \to G$ satisfies *a condition c_C over C inductively:*

- *Every morphism q satisfies $true$.*
- *A morphism q satisfies $\exists(a, c_D)$, denoted $q \models \exists(a, c_D)$, if there exists a monomorphism $q' : D \to G$ such that $q' \circ a = q$ and $q' \models c_D$.*
- *A morphism q satisfies $\neg c_C$ if it does not satisfy c_C and satisfies $\wedge_{i \in I} c_{C,i}$ if it satisfies each $c_{C,i}$ $(i \in I)$.*

For example, the property in Fig. 1, would be denoted by the nested condition over the empty graph $\exists(i_{C_1} : \emptyset \to C_1, \neg\exists(h_1 : C_1 \to C_2, \neg\exists(h_2 : C_2 \to C_3, true)))$ with nesting level 3. As said above, nested conditions are more general than needed. The graph properties in our graph logic *GL* are not arbitrary conditions, but conditions over the empty graph. Consequently in this case, the models of a condition are morphisms $\emptyset \to G$ and this is equivalent to say that models of these conditions are graphs G. However, we must notice that if $\exists(a, c)$ is a graph property, in general c is an arbitrary graph condition.

Definition 2 (GL Syntax, GL Semantics). *The language of graph properties GL consists of all conditions over the empty graph $\emptyset$. Given an element $\exists(a, c_D)$ of GL with $a : \emptyset \to D$, we also denote it by $\exists(D, c_D)$. A graph G satisfies a graph property c of GL if the unique morphism $i : \emptyset \to G$ satisfies c.*

2.2 Conjunctive Normal Form and Shifting Results

In this section, we introduce the notion of clause and (weak) conjunctive normal form in GL that is needed in the following section to present tableau reasoning [7] for GL.

Definition 3 ((weak) CNF, subcondition). *A* literal ℓ *is either a condition of the form* $\exists(a,d)$ *or* $\neg\exists(a,d)$*. We say that a literal of the form* $\exists(a,d)$ *is a* positive literal *and of the form* $\neg\exists(a,d)$ *is a* negative literal*. Each disjunction of literals is also called a* clause*. A condition is in* weak conjunctive normal form (CNF) *if it is either true, or false, or a conjunction of clauses* $\wedge_{j\in J} c_j$*, with* $c_j = \vee_{k\in K_j}\ell_k$*, where for each literal,* $\ell_k = \exists(a_k,d_k)$ *or* $\ell_k = \neg\exists(a_k,d_k)$*,* d_k *is a condition in weak CNF again. Moreover, for each negative literal* $\ell_k = \neg\exists(a_k,d_k)$ *it holds that* a_k *is a monomorphism, but not an isomorphism. A condition c is in* CNF *if it is in weak CNF and in addition for each literal* $\exists(a,d)$ *or* $\neg\exists(a,d)$ *occurring in c it holds that a is a monomorphism, but not an isomorphism. A* subcondition *of a condition c in (weak) CNF is either a clause or a conjunction of clauses in c.*

The idea of having conditions in weak CNF is to be able to handle them efficiently in our tableau reasoning method. In particular, the conjunction of clauses will allow us to apply the classical tableau rules for CNF formulas. In addition, since negative literals $\neg\exists(a,d)$ where a is not an isomorphism are always satisfiable, we can handle them in a specialized way as will be explained in the following section.

In [13,14], Pennemann describes a procedure, based on some equivalences, for transforming any condition into CNF, which makes it sufficient to formulate our tableau reasoning method based on nested tableaux (see Section 3.2) for graph properties in CNF. It is routine to show, based on the same equivalences, that the transformation to CNF does not increase the nesting level of conditions.

Lemma 1 (transformation to CNF [13,14]). *There exists a transformation for each condition* c_C *over C according to equivalences listed in [13,14] into an equivalent condition* $[c_C]$ *in CNF.*

Lemma 2 (transformation to CNF preserves or reduces nesting level). *Given a condition* c_C *over C with nesting level n,* $[c_C]$ *has nesting level not greater than n.*

In the following sections we will make use extensively of the following shifting result over morphisms for conditions [5] that allow us to move a condition along a morphism.

Lemma 3 (shift of conditions over morphisms). *There is a transformation Shift such that for each condition* c_P *over P and each morphism* $b : P \to P'$*, it returns a condition* $Shift(b,c_P)$ *over* P' *such that for each monomorphism* $n : P' \to H$ *it holds that* $n \circ b \models c_P \Leftrightarrow n \models Shift(b,c_P)$*.*

Construction 1 (shift of conditions over morphisms). *The transformation Shift is inductively defined as follows:*

$$\begin{array}{ccc} P & \xrightarrow{\ b\ } & P' \\ {\scriptstyle a}\downarrow & (1) & \downarrow{\scriptstyle a'} \\ C & \xrightarrow[\ b'\ ]{} & C' \\ \triangle & & \\ {\scriptstyle c_C} & & \end{array}$$

$Shift(b,true) = true.$

$Shift(b,\exists(a,c_C)) = \bigvee_{(a',b')\in\mathcal{F}} \exists(a',Shift(b',c_C))$ *if*

$\mathcal{F} = \{(a',b') \text{ jointly surjective} \mid b' \text{ mono and (1) commutes}\} \neq \emptyset$

$Shift(b,\exists(a,c_C)) = false$ *if* $\mathcal{F} = \emptyset$.

Moreover, $Shift(b,\neg c_P) = \neg Shift(b,c_P)$ *and*

$Shift(b,\wedge_{i\in I} c_{P,i}) = \wedge_{i\in I} Shift(b,c_{P,i})$.

When shifting conditions over morphisms we need to know if their nesting level is preserved/reduced and if they remain in weak CNF. It is routine to prove the lemma below by induction on the structure of the given condition. But shifting negative literals does not preserve weak CNF. When shifting a negative literal, according to Construction 1 we obtain $Shift(b, \neg\exists(a, c_C)) = \neg Shift(b, \exists(a, c_C) = \neg \bigvee_{(a',b')\in\mathcal{F}} \exists(a', Shift(b', c_C)) = \bigwedge_{(a',b')\in\mathcal{F}} \neg\exists(a', Shift(b', c_C))$. Notice that we require that a' is a monomorphismIn, which is not required in [5]. We may add this requirement because the models of our conditions are monomorphisms instead of arbitrary morphisms.

Lemma 4 (Shift literal in weak CNF). *Given a positive literal ℓ in weak CNF with nesting level n and a morphism $b : P \to P'$, $Shift(b, \ell)$ is a condition in weak CNF again and it has nesting level less than or equal to n. Given a negative literal ℓ in weak CNF with nesting level n and a morphism $b : P \to P'$, $Shift(b, \ell)$ has nesting level less than or equal to n.*

3 Tableau-Based Reasoning for GL

Analogously to tableaux for plain first-order logic reasoning [7], we introduce tableaux for dedicated automated reasoning for graph properties. We consider *clause tableau reasoning* assuming that the given graph conditions are in weak CNF as introduced in Def. 3. However, we will see that usual tableau reasoning is not sufficient and we will introduce so-called nested tableaux for automated graph property reasoning.

3.1 Tableaux for Graph Conditions

As usual, tableaux are trees whose nodes are literals. The construction of a tableau for a condition c_C in weak CNF can be informally explained as follows. We start with a tableau consisting of the single node *true*. Then, as usual in first-order logic, for every clause $c_1 \vee \ldots \vee c_n$ in c_C we extend all the leaves in the tableau with n branches, one for each condition c_i, as depicted in Fig. 2. The tableau rules that are specific for our logic are the so-called *lift* and *supporting lift* rules, defined by Penneman as part of his proof system [13,14]. In both rules, given two literals $\ell_1 = \exists(a_1, c_1)$ and ℓ_2 in the same branch, we add a new node to that branch with a new literal ℓ_3 that is equivalent to the conjunction of ℓ_1 and ℓ_2. In particular, ℓ_3 is built by pushing ℓ_2 inside the next level of nesting of ℓ_1 by shifting ℓ_2 (see Lemma 3) along a_1. The difference between the lift and supporting lift rules is that, in the former, ℓ_2 is a negative literal while, in the latter, ℓ_2 is a positive literal. Moreover, since at any time in the tableau we want to produce conditions that are in weak CNF and as argued in Lemma 4 shifted negative literals in general do not preserve weak CNF, we apply the transformation rules according to Lemma 1 to each shifted negative literal. The two rules are depicted in Fig. 3. Note that because of Lemma 4 and 1, the tableau rules and in particular the lift and supporting lift rule indeed generate a tableau as given in Def. 4, where each node is a literal in weak CNF. As usual, a closed branch is then a branch where we have found an inconsistency.

Definition 4 (Tableau, branch). *A* tableau *is a finitely branching tree whose nodes are literals in weak CNF (see Def. 1). A* branch *in a tableau T is a maximal path in T.*

Definition 5 (Tableau rules). *Given a condition c_C in weak CNF over C, then a tableau for c_C is defined as a tableau constructed with the following rules:*

- *The tree consisting of a single node* true *is a tableau for c_C (initialization rule).*
- *Let T be a tableau for c_C, B a branch of T, and $c_1 \vee \ldots \vee c_n$ a clause in c_C, then extend B with n new subtrees and the nodes of the new subtrees labeled with c_i (extension rule).*
- *Let T be a tableau for c_C, B a branch of T, and $\exists(a_1, c_1)$ and $\neg\exists(a_2, c_2)$ literals in B, then extend B with a node equal to $\exists(a_3, c_3)$, where $\exists(a_3, c_3)$ is the condition $\exists(a_1, c_1 \wedge [Shift(a_1, \neg\exists(a_2, c_2))])$ (*lift rule*).*
- *Let T be a tableau for c_C, B a branch of T, and $\exists(a_1, c_1)$ and $\exists(a_2, c_2)$ nodes in B, then extend B with a node equal to $\exists(a_3, c_3)$, where $\exists(a_3, c_3)$ is the condition $\exists(a_1, c_1 \wedge Shift(a_1, \exists(a_2, c_2)))$ (*supporting lift rule*).*

Definition 6 (Open/closed branch). *In a tableau T a branch B is* closed *if B contains $\exists(a, false)$ or $false$; otherwise, it is* open.

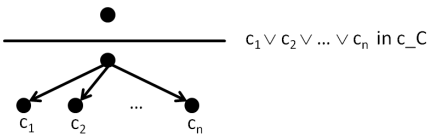


Fig. 2. The extension rule

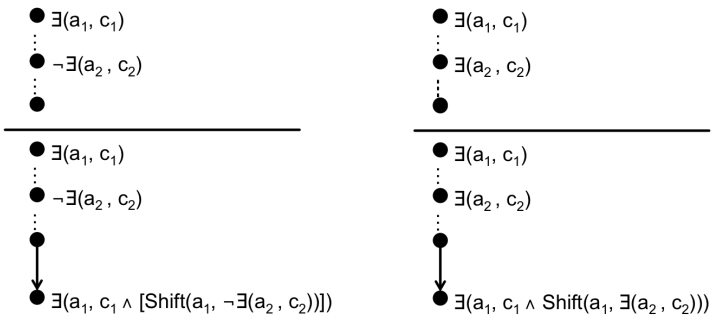


Fig. 3. The lift rule (left) and supporting lift rule (right)

In the completeness proof for our tableau reasoning method (see Theorem 2) we will need to track conditions that, because of the lift and supporting lift rules, move inside other conditions. We do this by means of a successor relation between conditions (and subconditions) for each branch in a given tableau.

Definition 7 (Successor relation for a branch). *For every branch B in a given tableau T, we define the successor relation associated to that branch as the least relation on nested conditions satisfying:*

- *If* $\exists(a_1, c_1 \wedge [Shift(a_1, \neg\exists(a_2, c_2))])$ *is the literal on a node created using the lift rule from literals* $\exists(a_1, c_1)$ *and* $\neg\exists(a_2, c_2)$*, then* $[Shift(a_1, \neg\exists(a_2, c_2))]$ *is a successor of* $\neg\exists(a_2, c_2)$ *in B.*
- *If* $\exists(a_1, c_1 \wedge Shift(a_1, \exists(a_2, c_2)))$ *is the literal on a node created using the supporting lift rule from literals* $\exists(a_1, c_1)$ *and* $\exists(a_2, c_2)$*, then* $Shift(a_1, \exists(a_2, c_2))$ *is a successor of* $\exists(a_2, c_2)$ *in B.*

Let us see an example that shows what happens if we have a very obvious refutable condition $\exists(a, true) \wedge \neg\exists(a, true)$. Since we have two clauses consisting of one literal, using the extension rule we would get a single branch with the literals $\exists(a, true)$ and $\neg\exists(a, true)$, where a is not an isomorphism. Using the lift rule we would extend the branch with the condition $\exists(a, true \wedge [Shift(a, \neg\exists(a, true))])$, which is equivalent to $\exists(a, true \wedge [\neg Shift(a, \exists(a, true))])$.

$$\begin{array}{ccc} P & \xrightarrow{a} & C \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle id} \\ C & \xrightarrow[id]{} & C \end{array}$$

According to the construction of $Shift$, $Shift(a, \exists(a, true))$ is a disjunction that would include $\exists(id, true)$, since the diagram on the left commutes, the identity is a monomorphism and (id, id) are jointly surjective. Therefore, the condition $[\neg Shift(a, \exists(a, true))]$ would be equal to $false$. The lift rule has thus created a literal $\exists(a, true \wedge false)$ on the branch manifesting the obvious contradiction between the literals $\exists(a, true)$ and $\neg\exists(a, true)$ in the inner condition of the literal $\exists(a, true \wedge false)$.

Using the lift and supporting lift rules we can thus make explicit the inconsistencies that occur at the outer level of nesting, but we would need additional tableau rules that do something similar at any inner level of nesting. The problem is that, then, it is very difficult to use induction to prove properties of these tableaux (and, in particular, completeness). Instead, in our procedure, after applying the extension rule until no new literals can be added, if the literals in the branch are $\ell_1, \ldots, \ell_n$ we choose a positive literal [2], say ℓ_1, that we call the *hook* of the branch, and we apply a lift (or a supporting lift) rule to ℓ_1 and ℓ_2. Then, if more literals are left, we apply again a lift rule to the result and ℓ_3 and so on, until we have applied a lift or supporting lift rule to all the literals in the branch or until the branch is closed. In the example described above, where we have a positive and negative literal on a single branch we would have the hook $\exists(a, true)$ and the literal $\neg\exists(a, true)$ was shifted by the lift rule to the inner condition of this hook. When we have done this for all the branches, then we say that the tableau is *semi-saturated*. The next step is that for every open branch, if the condition in the leaf is $\exists(a, c)$, then we will proceed to *open* a new tableau for the inner condition c, trying to find a refutation for c. In our example, the leaf of the open branch is equal to $\exists(a, true \wedge [\neg Shift(a, \exists(a, true))]) = \exists(a, true \wedge false)$ and we would therefore open a new tableau for $true \wedge false$. It is obvious that the new tableau will be closed and

[2] If all the literals are negative, then no rule can be applied. But in this case, we can conclude that the given condition c_C is satisfiable. The reason is that, if c_C is a condition over C, the identity $id : C \to C$ would be a model for all the literals in the path and, hence, for the condition.

will therefore refute the inner condition of $\exists(a, true \wedge false)$ making also the original condition $\exists(a, true) \wedge \neg\exists(a, true)$ refutable. If in the newly opened tableaux there are still some open branches however, then new tableaux will be created for the conditions in the leaves, and so on. We call such a family of tableaux a *nested tableau* and we will study them in the following section.

Definition 8 (Semi-saturation, hook for a branch). *Given a* tableau *T for a condition c_C over C, we say that T is* semi-saturated *if*

- *No new literals can be added to any branch B in T using the extension rule.*
- *And in addition, for each branch B in T, one of the following conditions hold*
  - *B is closed,*
  - *B consists only of negative literals,*
  - *If $E = \{\ell_1, \ldots, \ell_n\}$ is the set of literals added to B using the extension rule, then there is a positive literal $\ell = \exists(a, d)$ in E such that the literal in the leaf of B is $\exists(a, d \wedge_{\ell' \in (E \setminus \{\ell\})} Shift(a, \ell'))$. Then, we say that l is a* hook *for branch B in T.*

For any condition in weak CNF we can then build a finite semi-saturated tableau.

Lemma 5 (Finiteness and existence of semi-saturated tableau). *Given a condition c_C over C in weak CNF, there exists a semi-saturated and finite tableau T for c_C.*

In the rest of this subsection we relate satisfiability of conditions in weak CNF with satisfiability of their associated tableaux. First, we define in the obvious way tableau satisfiability. Then, we show that if a condition c_C is satisfiable, then any associated tableau cannot have all its branches closed. This means that the tableau rules are sound. The proof is by induction on the structure of the tableau. The base case is trivial. If a node has been added by using the extension rule, then satisfiability of the given condition implies satisfiability of the tableau. Finally, the case of the lift and the supporting lift rules is a consequence of the soundness of these rules as shown by Penneman [13].

Definition 9 (Branch and tableau satisfiability). *A* branch *B in a tableau T for a condition c_C over C is* satisfiable *if there exists a morphism $q : C \to G$ satisfying all the literals in B. In this case we say that $q : C \to G$ is a model for B and we write $q \models B$. A* tableau *T is* satisfiable *if there is a satisfiable branch B in T. If $q \models B$ in T, we also say that q is a model for T and also $q \models T$.*

Lemma 6 (Tableau soundness). *Given a condition c_C in weak CNF and a tableau T for this condition, then if c_C is satisfiable, so is T.*

Now, we show that if a tableau T is semi-saturated and satisfiable, then its associated condition c_C is also satisfiable. Actually, we show something slightly stronger. In particular, if an open branch B in T includes positive literals and the literal in the leaf is satisfiable then c_C is also satisfiable. The first part of the proof of this lemma is trivial, since semi-saturation ensures that each branch includes a literal for each clause in c_C. The second part uses the fact that semi-saturation implies that d includes the shift of all the literals in the branch (except for the hook). Then, satisfaction of this condition implies the satisfaction of all conditions in the branch.

Lemma 7 (Semi-saturation and satisfiability). *If T is a semi-saturated tableau for c_C, then $q \models T$ implies $q \models c_C$. Moreover, if B is an open branch including some positive literals and $\exists(a, d)$ is the literal in the leaf of B, then $q \models \exists(a, d)$ implies $q \models c_C$.*

3.2 Nested Tableaux for Graph Properties

As we have discussed in the previous section, standard tableaux cannot be used in a reasonably simple way as a proof procedure in our logic. So, our approach is based on a notion of *nested tableaux* whose idea is that, for each open branch of a tableau T whose literal in the leaf is $\exists(a,c)$, we open a new tableau T' to try to refute condition c. Then, we say that $\exists(a,c)$ is the *opener* for T' and if a is a morphism from G to G', we say that G' is the context of T', meaning that c is a property of the graph G'.

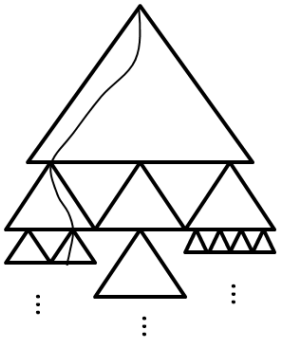


Fig. 4. Nested tableau with nested branch

Nested tableaux have nested branches consisting of sequences of branches of a sequence of tableaux in the given nested tableau. We may notice that while our tableaux are assumed to be finite (cf. Lemma 5), nested tableaux and nested branches may be infinite. For simplicity, we will assume that the original condition to (dis)prove is a graph property, i.e. the models of the given condition (if they exist) are intuitively graphs. Formally, this means that the given condition c is a condition over the empty graph, i.e. on the outer level of nesting it consists of literals of the form $\exists(a,d)$ or $\neg\exists(a,d)$, where a is some morphism over the empty graph. Similarly, the models of c are morphisms from the empty graph.

Definition 10 (Nested tableau, opener, context, nested branch, semi-saturation). *Let $(I,\leq,i_0)$ be a poset with minimal element i_0. A* nested tableau *NT is a family of triples $\{\langle T_i,j,\exists(a_j,c_j)\rangle\}_{i\in I}$, where T_i is a tableau and $\exists(a_j,c_j)$, called the* opener *of T_i is the literal of an open branch in T_j with $j<i$. Moreover, we assume that there is a unique initial tableau T_{i_1} that is part of the triple $\langle T_{i_1},i_0,true\rangle$.*

We say that T_{i_1} has the empty context $\emptyset$ *and for any other tableau T_i, with $\langle T_i,j,\exists(a_j,c_j)\rangle\in NT$, if $a_j:A_j\to A_{j+1}$, we say that T_i has* context A_{j+1}.

A nested branch NB *in a nested tableau $NT=\{\langle T_i,j,\exists(a_j,c_j)\rangle\}_{i\in I}$ is a maximal sequence of branches $B_{i_1},\ldots,B_{i_k},B_{i_{k+1}},\ldots$ from 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}, such that if B_{i_k} and $B_{i_{k+1}}$ are two consecutive branches in the sequence then the leaf in B_{i_k} is the opener for $T_{i_{k+1}}$.*

Finally, NT is semi-saturated *if each tableau in NT is semi-saturated.*

Definition 11 (Nested tableau rules). *Given a graph property c in CNF, a* nested tableau *for c is constructed with the following rules:*

- *Let T_{i_1} for c be a tableau constructed according to the rules given in Def. 5, then $\{\langle T_{i_1},i_0,true\rangle\}$ is a nested tableau for c (*initialization rule*).*
- *Let $NT=\{\langle T_i,j,\exists(a_j,c_j)\rangle\}_{i\in I}$ be a nested tableau for c and $\exists(a_n,c_n)$ with $a_n:A_n\to A_{n+1}$ a literal in a leaf of a tableau T_n in NT such that $\exists(a_n,c_n)$ is not the opener for any other tableau in NT, then add to NT a triple $\langle T_j,n,\exists(a_n,c_n)\rangle$, where $j>n$ is an index not previously used in I of NT, and T_j is a tableau for c_n (*nesting rule*).*

For the proof of the completeness theorem, we need to extend the successor relation defined for the branches of a tableau to a successor relation defined for each nested branch of a nested tableau.

Definition 12 (Successor relation for a nested branch). *For each nested branch NB in a nested tableau NT, we define the successor relation associated to that branch as the least transitive relation on nested conditions satisfying:*

- *If c_1 is a successor of c_2 in a branch B, with context A, in NB, then $\langle A, c_1\rangle$ is the successor of $\langle A, c_2\rangle$ in NB*
- *If $\exists(a_n, c_n)$ with $a_n : A_n \to A_{n+1}$ is the opener for a branch B in NB and ℓ is a literal included in B using the extension rule, then $\langle A_{n+1}, \ell\rangle$ is the successor of $\langle A_n, c\rangle$ for each condition c that is a subcondition of c_n.*

If the context of conditions is clear from the context, then we may just say that a condition c is a successor of c' in NB, instead of saying that $\langle A, c\rangle$ is a successor of $\langle A', c'\rangle$ in NB, where A and A' are the contexts of c and c' respectively.

A result that will be needed in the completeness proof is the fact that, if all the successors of a literal in a given branch are satisfiable, then the literal is satisfiable. The proof is by induction on $j - i$, using the definition of the Shift operation and of the successor relation, and the fact that all the tableaux involved are semi-saturated.

Lemma 8 (Satisfiability of successors). *Let $NB = B_0, B_1, \ldots, B_j, \ldots$ be a nested branch in a semi-saturated nested tableau, ℓ_i a literal in B_i with context A_i, $a_{ij} : A_i \to A_j$ the composition $a_{j-1} \circ \cdots \circ a_i$. Then if $\langle A_j, \ell_j\rangle$ is the successor of ℓ_i in B_i and $q_j \models \ell_j$, then $q_j \circ a_{ij} \models \ell_i$.*

As in the case of standard tableaux, a closed nested branch represents an inconsistency detected between the literals in the branch, and an open branch represents, under adequate assumptions, a model of the original condition.

Definition 13 (Open/closed nested branch, nested tableau proof). *A nested branch NB in a nested tableau NT for a graph property c in CNF of GL is* closed *if NB contains $\exists(a, false)$ or $false$; otherwise, it is* open*. A nested tableau is closed if* all *its nested branches are closed.*

A nested tableau proof *for (the unsatisfiability of) c is a closed nested tableau NT for c in CNF of GL according to the rules given in Def. 11.*

Example 1. Let us consider another simple example of deduction with our tableau method. Suppose that we want to resolve the condition c:
$\exists(\emptyset \to Node, true) \wedge \neg\exists(\emptyset \to Loop, true) \wedge \neg\exists(\emptyset \to Node, \neg\exists(Node \to Loop, true))$
Thereby, $Node$ is a graph consisting of just one node, and $Loop$ is a graph consisting of a node and a loop. That is, c states (i) that there must exist a node, (ii) there cannot be loops and (iii) there is no node that does not include a loop. The tableau associated to c is depicted in Fig. 5. It includes just a branch, since there are no disjunctions in c. The first nodes of this branch include three literals obtained by applying the extension rule. Moreover, the first literal of these three nodes is the hook for the branch, since it is the only positive literal. Then, after semi-saturation, the leaf of the branch would be $\exists(\emptyset \to Node, d)$, where d is depicted at the bottom of the figure.

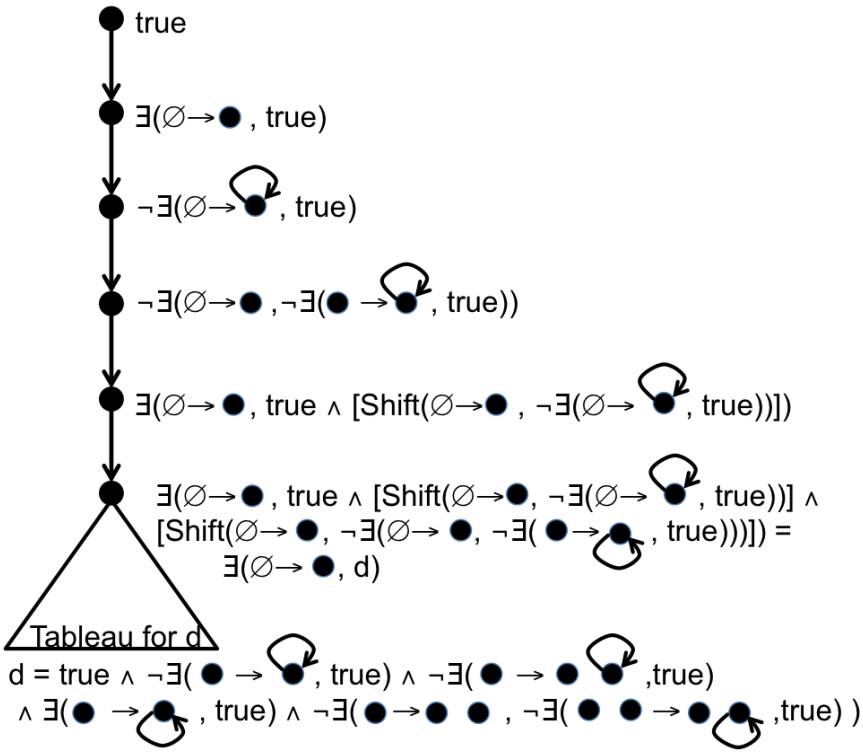


Fig. 5. Tableau for condition c in Example 1

So, the tableau for d with context $Node$ is depicted in Fig. 6. Again, it includes a single branch whose hook is the only positive literal in d. In this case, the leaf of the branch would be $\exists(Node \to Loop, d')$, where d' is equivalent to the conjunction of shifting along $Node \to Loop$ with the rest of the literals in d. But in this case, because of shifting the negative literal $\neg\exists(Node \to Loop, true)$ the conjunction d' would include the $false$ literal: As a consequence, when opening a tableau for d' at the next level of nesting, the only branch would include the $false$ literal, which would close the single branch. Hence, this nested tableau would be a proof of the unsatisfiability of c.

4 Soundness and Completeness

In this section we prove that our tableau method is sound and complete. In particular, soundness means that if we are able to construct a nested tableau where all its branches are closed then we may be sure that our original condition c is unsatisfiable. Completeness means that if a *saturated* tableau includes an open branch, where the notion of saturation is defined below, then the original condition is satisfiable. Actually, the open branch provides the model that satisfies the condition.

Theorem 1 (Soundness). *If there is a nested tableau proof for the graph property c in CNF of GL, then c is unsatisfiable.*

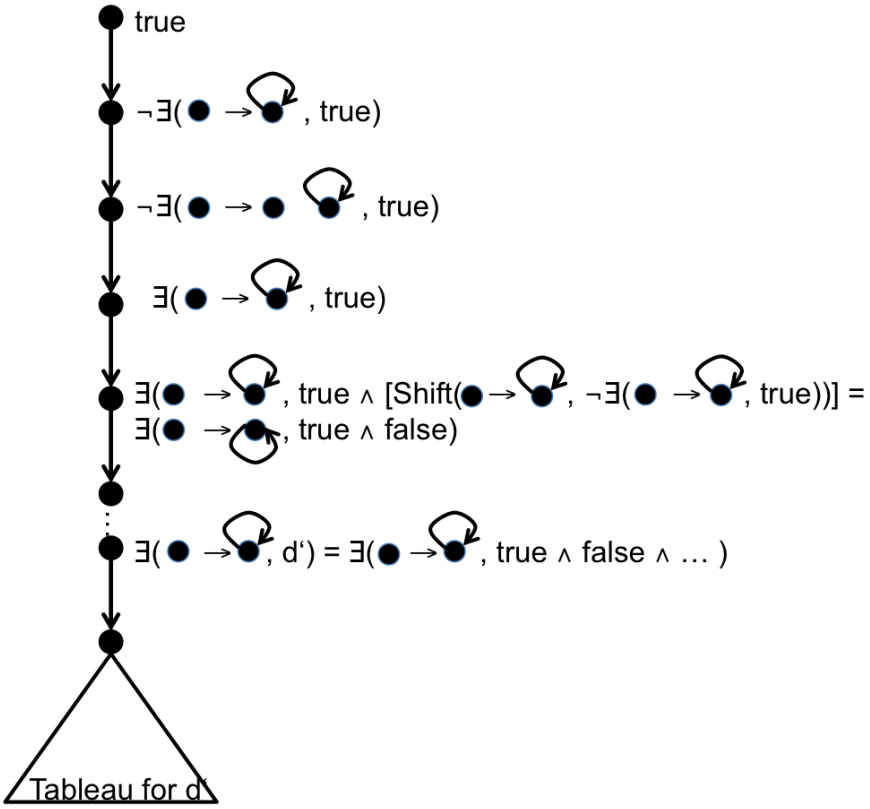


Fig. 6. Tableau for condition *d* in Example 1

In the proof of this theorem we use Lemma 6 that states the soundness of the rules for constructing (non-nested) tableaux and the fact that if all branches of the nested tableau are closed then it is finite. In particular, we prove by induction on the structure of *NT* that if *c* is satisfiable, then it must include an open branch. The base case is a consequence of Lemma 6. For the general case, assuming that the given nested tableau NT_i has an open nested branch *NB*, we consider two cases, depending on how we extend that NT_i. If the new tableau is not opened at the leaf of *NB*, then *NB* is still an open branch of the new tableau. Otherwise, we show that *NB* can be extended by a branch of the new tableau using Lemma 6.

For the completeness proof, the notion of saturation of nested tableaux is important. As usual, saturation describes some kind of fairness that ensures that we do not postpone indefinitely some inference step. In this case, the main issue concerning fairness is the choice of the hook for each tableau in the given nested tableau. In particular, if a (positive) literal, or its successors are never chosen as hooks we will be unable to make inferences between that literal and other literals, especially, negative literals. This means that we may be unable to find some existing contradictions.

Definition 14 (Saturation). *A nested tableau NT is* saturated *if the following conditions hold:*

1. *NT is semi-saturated.*
2. *For each open nested branch $NB = B_0, B_1, \ldots, B_j, \ldots$ in NT one of the following conditions hold:*
   - *NB includes a branch consisting only of negative literals.*
   - *For each node $\exists(a_j, c_j)$ in NB with nesting level k on branch B_j in tableau T_j either $\exists(a_j, c_j)$ is a hook for B_j and the leaf of B_j is a tableau opener* (positive literal is hook)*, or there is a successor $\exists(a_i, c_i)$ of $\exists(a_j, c_j)$ with nesting level k being a hook in branch B_i of NB with $i > j$ and corresponding tableau opener* (positive literal is a hook in the future).

Lemma 9 (Existence saturated nested tableau). *Given a graph property c in CNF of GL, then there exists a saturated nested tableau NT for c.*

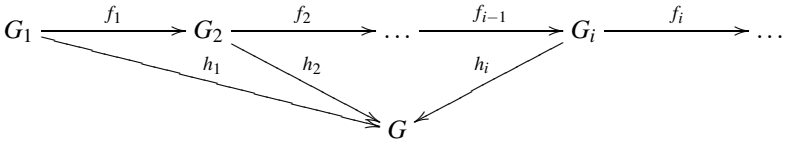
The key issue here is to choose the adequate hook for each branch B_i in each nested branch $NB = B_0, B_1, \ldots, B_j, \ldots$. This can be done by keeping, for each branch, a queue of pending literals. So when choosing the hook for branch B_i, if the first literal in the queue is $\exists(a_j, c_j)$, we select the successor of $\exists(a_j, c_j)$ in B_i.

To prove the completeness theorem we will show that, to any open branch in a given nested tableau NT, we can associate a graph G that, if NT is saturated, will be a model for NT. In particular, it is defined as the colimit of monomorphisms arising from the sequence of tableau openers on the nested branch. The existence of infinite colimits (satisfying an additional minimality property) is described in Proposition 1, which is proven in [12]. Intuitively, these colimits are the union of the graphs in the sequence.

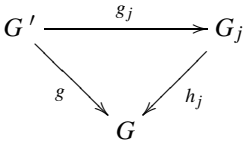
Proposition 1 (Infinite colimits). *Given a sequence of monomorphisms:*

$$G_1 \xrightarrow{f_1} G_2 \xrightarrow{f_2} \ldots \xrightarrow{f_{i-1}} G_i \xrightarrow{f_i} \ldots$$

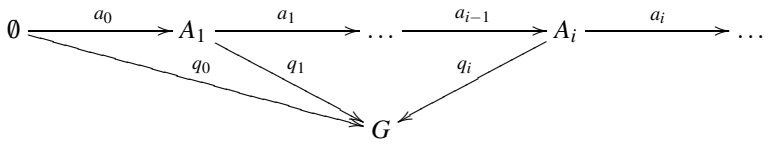
there exists a colimit:



that satisfies that for every monomorphism $g : G' \to G$, such that G' is a finite graph, there is a j and a monomorphism $g_j : G' \to G_j$ such that the diagram below commutes:



Definition 15 (Canonical model for an open nested branch *NB*). *Given a nested tableau NT for c in CNF of GL and given an open nested branch NB in NT, then the canonical model for NB is the empty graph in case that $NB = B_{i_1}$ consists of only one branch and otherwise it is defined as the (possibly infinite) colimit*



from the sequence of monomorphisms $\emptyset \overset{a_0}{\to} A_1 \overset{a_1}{\to} A_2 \cdots \overset{a_{i-1}}{\to} A_i \overset{a_i}{\to} A_{i+1} \cdots$ arising from the sequence of tableau openers true, $\exists(a_0,c_0)$, $\exists(a_1,c_1)$, $\cdots$, $\exists(a_i,c_i)\cdots$ on the nested branch NB in NT.

Theorem 2 (Completeness). *If the graph property c in CNF of GL is unsatisfiable, then there is a tableau proof for c.*

In the proof of the Completeness Theorem we show that if there is no tableau proof for c, then c is satisfiable. Because of Lemma 9 we know that there exists a saturated nested tableau NT for c. If NT is not a tableau proof for c, there exists at least one open nested branch NB in NT. Then, we prove by induction on the nesting level of literals that its associated canonical model G (as given in Def. 15) satisfies all the literals in NB. In particular, we show that morphism $q_n : A_n \to G$ satisfies all literals whose context is A_n. This means that each condition on the branch B_{i_1} with empty context within NB is satisfied by $q_0 : \emptyset \to G$ and thus by G. Hence, as a consequence of semi-saturation and Lemma 7, it holds that $G \models c$.

5 Conclusion

In this paper, we have presented a new tableau-based reasoning method for graph properties, where graph properties are formulated explicitly making use of graphs and graph morphisms, having the expressive power of Courcelle's first order logic of graphs. In particular, we proved the soundness and completeness of this method based on Pennemann's [13] rules. Finally, we presented a new notion of nested tableaux that could be useful for other formalisms where formulas have a hierarchical structure like nested graph conditions. With respect to future work, in addition to implementing our approach, we consider that the graph logic GL could be used as the foundations for graph databases, in the same sense as first-order logic is used as foundations for relational databases. In particular there would be two aspects that would need further work. On the one hand, we would need to extend the logic and reasoning method to allow for the possibility to state the existence of paths between nodes in a graph. On the other hand, we would need to characterize a sufficiently expressive fragment of that logic that is not only decidable, but where queries could be computed efficiently.

Acknowledgement. We would like to thank Annegret Habel and Karl-Heinz Pennemann for some interesting discussions on the main ideas of this paper.

References

1. Boronat, A., Meseguer, J.: Automated model synchronization: A case study on uml with maude. ECEASST 41 (2011)
2. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
3. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars, pp. 313–400. World Scientific (1997)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. Springer (2006)
5. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: $\mathcal{M}$-adhesive transformation systems with nested application conditions. part 1: Parallelism, concurrency and amalgamation. Math. Struct, in Comp. Sc. (2012) (to appear)
6. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19(2), 245–296 (2009)
7. Hähnle, R.: Tableaux and related methods. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 100–178. Elsevier and MIT Press (2001)
8. Heckel, R., Wagner, A.: Ensuring consistency of conditional graph rewriting - a constructive approach. Electr. Notes Theor. Comput. Sci. 2, 118–126 (1995)
9. Lack, S., Sobocinski, P.: Adhesive and quasiadhesive categories. ITA 39(3), 511–545 (2005)
10. Lambers, L., Orejas, F.: Tableau-based reasoning for graph properties. Tech. rep., Departament de Llenguatges i Sistèmes Informàtics, Universitat Politècnica de Catalunya (2014)
11. Orejas, F.: Symbolic graphs for attributed graph constraints. J. Symb. Comput. 46(3), 294–315 (2011)
12. Orejas, F., Ehrig, H., Prange, U.: Reasoning with graph constraints. Formal Asp. Comput. 22(3-4), 385–422 (2010)
13. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 289–304. Springer, Heidelberg (2008)
14. Pennemann, K.H.: Development of Correct Graph Transformation Systems, PhD Thesis. Dept. Informatik, Univ. Oldedburg (2009)
15. Rensink, A.: Representing first-order logic using graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004)