A Navigational Logic for Reasoning about Graph Properties[☆]

Marisa Navarro[a], Fernando Orejas[b], Elvira Pino[b], Leen Lambers[c]

[a] *Universidad del País Vasco (UPV/EHU), San Sebastián, Spain*
[b] *Universitat Politècnica de Catalunya, Barcelona, Spain*
[c] *Hasso Plattner Institut, University of Potsdam, Germany*

Abstract

Graphs play an important role in many areas of Computer Science. For this reason, it is very important to have the means to express and to reason about the properties that a given graph may satisfy. In particular, our work is motivated by model-driven software development and by graph data bases. With this aim, in this paper we present a visual logic that allows us to describe graph properties, including *navigational* properties, i.e., properties about the paths in a graph. The logic is equipped with a deductive tableau method that we have proved to be sound and complete.

Keywords: Graph Logic, Algebraic Methods, Formal Modelling, Specification

1. Introduction

Being able to state properties about graphs and to reason about them is important in many areas of computer science where graphs play a relevant role. For instance, in software and system modeling, where models are described using different graphical notations, graph properties may be used to describe properties of the given models, and reasoning tools may be used for model validation. Similarly, in the context of graph databases, graph properties could be used to express integrity constraints or just to express queries to the database. In that context, reasoning tools may allow us to check these constraints or to validate the search engine to satisfy these queries.

Two kinds of approaches can be used to describe graph properties. The first one is based on using some standard logic, after encoding some graph concepts into it. For instance, Courcelle (e.g., [1]) studied a graph logic defined in terms of first-order (or monadic second-order) logic, extended with a predicate $node(n)$ for stating that n is a node, and with a predicate $edge(n,n')$, for stating that there is an edge from node n to n'. A similar logic, but with some important differences is presented in

[☆] Funding: This work was supported by the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant GRAMM (ref. TIN2017-86727-C2-1-R, TIN2017-86727-C2-2-R) and by the Basque Project GIU15/30, and grant UFI11/45.

Email addresses: marisa.navarro@ehu.es (Marisa Navarro), orejas@cs.upc.edu (Fernando Orejas), pino@cs.upc.edu (Elvira Pino), Leen.Lambers@hpi.de (Leen Lambers)

[2] by Cardelli, Gardner and Ghelli whose expressive power is shown to be between first-order and monadic second-order logic. The second kind of approach is based on defining a dedicated logic, where graph properties may be expressed in terms of formulas that directly include graphs or graph concepts. The most important example in this direction is the *Logic of Nested Graph Conditions* (LNGC), introduced by Habel and Pennemann [3], which was proven to be equivalent to the first-order logic of graphs of Courcelle.

There are two main advantages in working with a logic like LNGC. The first one is efficiency. Pennemann [4] showed that a specialized prover for their logic outperformed some standard provers, like Darwin [5] or Vampire [6], when applied to graph formulas using Courcelle's logic. The second advantage is generality, in the sense that LNGC, being formulated in terms of category theory, can be used for any category of structures, as long as that category satisfies some general properties, i.e., LNGC can be used for stating properties and reasoning about many classes of graphs. This is important because there are many kinds of graphical structures that can be of interest in different areas of Computer Science. Conversely, using a standard logic, we need a different encoding for each class of structures. Moreover, specific results for one class may not be easy to transfer to another.

A main problem of LNGC is that it cannot be used to express *navigational* properties, i.e., path properties like "there is a path from node n to n'", which are important in many application areas, but are (monadic) second-order properties. In this sense, the main aim of this paper is the presentation of an extension of LNGC that allows us to state path properties and to reason about them.

This paper puts together, extends and complements previous work from the authors on graph logics, most of it on LNGC (i.e., without path properties) or some fragments of it. In particular, in [7] we presented the first version of our tableau method for reasoning in LNGC, showing its soundness and completeness, while in [8, 9] we presented a method for generating models of LNGC formulas. It is worth pointing out that our results show that our tool AutoGraph [9] can compete with respect to efficiency with a general tool like Alloy, thereby computing stronger results in the sense that AutoGraph generates minimally representable symbolic models (without a predefined scope as in Alloy). In [10] we dealt with paths for the first time, but in a very restrictive setting: we considered only Horn clause-like formulas over XML structures, i.e., trees. The problem was considerably simpler than the one approached in the current paper. First, because trees are simpler than graphs and, second, because we worked only with a fragment of LNGC. Finally, [11, 12] can be considered extensions of this paper, even if they have been published before. In particular, in [11] we described some preliminary ideas of how our approach could be generalized to arbitrary categories of structures, and in [12] we showed that that this new logic is an institution [13], providing means for structuring and modularizing specifications over it [14]. In both cases, the work was essentially semantical, working at the model level rather than at the deduction level.

The main contributions of this work are:

- The definition of GNL (Graph Navigational Logic), extending LNGC with path expressions, but using a notion of satisfaction, which is not the standard one, that considerably simplifies some technical constructions and, most importantly,

allows us to avoid exponential branching in our tableau proofs, which we would have if using the standard notion of satisfaction.

- The definition of a tableau method for reasoning in this logic, showing its soundness and completeness. The method uses nested tableaux, as in [7], with new tableau rules that are needed because of path expressions.

- As a by-product, we have that our new tableau method is also a sound and complete proof method for LNGC (with respect to our notion of satisfaction).

The paper is organized as follows. In Sect. 2 we introduce graphs, patterns and how they are related, and in Sect. 3 we introduce the syntax and semantics of GNL, including some basic results that are needed in the rest of the paper. In Sect. 4 we present our tableau reasoning method, and in Sect. 5 we show that the method is sound and we present our completeness results. In particular, we show that our method is complete for a class of graphs including infinite paths. In addition, we show that our tableau rules, excluding path unfolding, are complete for the class of graphs when formulas do not include path expressions. This can be seen as a reformulation of the results presented in [7], where only monomorphisms were used in formulas and in the satisfaction relation. In that new reformulation two additional tableau rules are needed but, conversely, the completeness proof is simpler, because of the notion of satisfaction used. In Sect. 6 we describe related work and in Sect. 7 we present some conclusions and we discuss how GNL could be used in the areas of Model Driven Development and as a logical foundation for graph databases. To enhance readability, the proofs of some technical intermediate results are just sketched, but in Appendix A we include all the missing details. To conclude, Appendix B shows a slightly more complex example than the examples used in the main text of our tableau reasoning method.

2. Basic Concepts

In this section, we present the basic concepts required to introduce GNL. In the first subsection we present a simple motivating example to provide some intuition on our constructions. Then, in the second subsection, we provide formal definitions for the main concepts underlying GNL and we show some of their properties.

2.1. An Example

Roughly, the idea of GNL (and of [3, 15]) is that basic properties state if a given pattern is present in a graph. In our case, patterns are graphs including some special edges (depicted as thick edges) that represent paths. Moreover, these paths are labelled with languages[1] over the set of edge labels. Then, when a pattern includes a path edge between two nodes labelled with L, that path edge describes a set of paths, each of them consisting of a sequence of edges, whose associated sequence of labels belongs to L. For this reason, this kind of path specifications are called *path expressions*.

[1] In practice, we would label these paths with some kind of expressions that denote a language, like regular expressions.

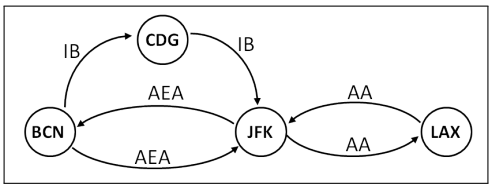


Figure 1: A graph of connected airports

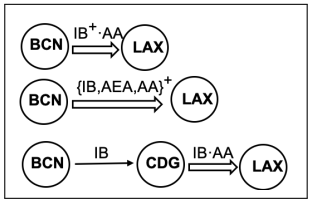


Figure 2: Three connection patterns

For instance, in Fig. 1 we depict a graph in the category of labelled graphs, including labels in nodes, that could be a fragment of a graph database, where nodes represent airports, edges represent direct flights, and edge labels represent the name of the company operating the flight.

Then, in Fig. 2, we depict three patterns for our airport network. The first one represents a connection from BCN to LAX consisting of a non-empty sequence of IB flights, followed by an AA flight, while the second one represents any connection (in terms of the three companies considered) from BCN to LAX. Notice that a connection labelled with $\{IB,AEA,AA\}^+$ means that every flight may be operated by any of the companies. The third pattern represents a direct IB flight from BCN to CDG followed by a connection from CDG to LAX consisting of an IB flight and an AA flight.

Then, formulas or graph conditions in GNL are built over patterns (and pattern morphisms) using quantifiers and the standard logical connectives. For instance, in Fig. 3 we present in a semi-formal notation some requirements on the airport network (in Sect. 3 we introduce the formal notation for graph conditions and Ex. 1 provides its formalization).

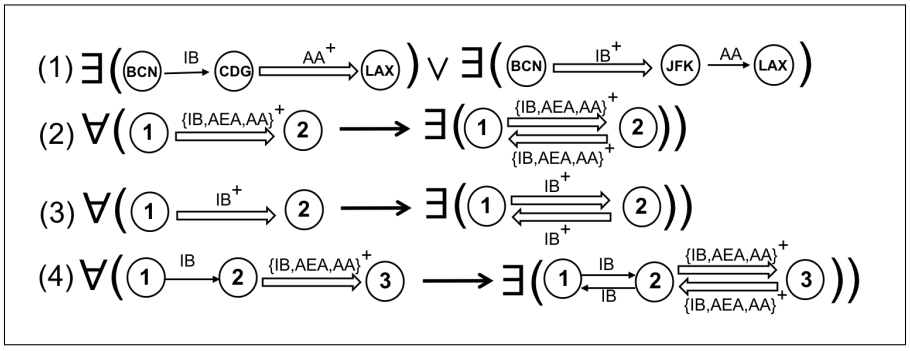


Figure 3: Properties on airport networks

The first graph condition states that there should be a connection from BCN to LAX consisting of an IB flight from BCN to CDG followed by a connection from CDG to LAX, in terms of AA flights, or a connection from BCN to JFK using IB flights followed by an AA flight from JFK to LAX. This condition may also represent a query of a customer that wants to fly from BCN to LAX and is willing to make a stopover either in CDG or in JFK such that, in the first case, the customer demands to fly directly to CDG with the company IB, whereas, in the second case, the requirement

is that the flight from JFK to LAX must be direct and operated by AA.

The second graph condition states that for every connection from an airport 1 to an airport 2, there must exist a backward connection from 2 to 1. Notice that 1 and 2 are not real labels, but metalabels used to identify nodes in the first pattern with nodes in the second pattern of the condition and the long arrow denotes a morphism (here an inclusion). The third condition is similar to the second one but, in this case, all the flights must be operated by IB. Finally, the fourth condition in Fig. 3 states that for each connection from an airport 1 to an airport 3 with a first stopover at an airport 2 by an IB flight, it must be possible to go back from 3 to 1 with a flight plan that also stops at 2, but as the last stopover before directly flying back to 1 with IB.

The last three conditions could be integrity constraints for the database. However, we may notice that our airport network in Fig. 1 does not satisfy the third requirement. For instance, there is a connection from BCN to JFK operated by IB, but there is no connection from JFK to BCN only operated by IB. The airport network in Fig. 1 neither satisfies the fourth requirement. For instance, there is a direct IB flight from BCN to CDG followed by a connection from CDG to LAX, but there is no connection from LAX to BCN whose last stopover is CDG.

2.2. Graph Patterns and Pattern Morphisms

In this subsection, we present the notions of graph patterns and graph pattern morphisms built over a category $\underline{\texttt{Graph}}$ of edge-labelled graphs, showing that they form the category $\underline{\texttt{Pattern}}$ that inherits from $\underline{\texttt{Graph}}$ some algebraic properties required to obtain the intended results.

Even if the aim of this work focuses on presenting a navigational logic for patterns based on a class of directed edge-labelled graphs as those used in the examples in the previous subsection, we believe that the results presented can be generalized to arbitrary classes of graphical structures, following ideas presented in [11]. Nevertheless, in order to be as general as possible, we only assume that categories consist of graphs equipped at least with a set of nodes and a set of directed labelled edges and, similarly, morphisms include at least a node mapping and a set mapping between the given sets of nodes and edges, respectively. This covers not only the category of labelled graphs, but also the categories of directed graphs (we only need to assume that a directed graph is a labelled graph where Σ consists of just one label) or of attributed graphs (in addition to directed labelled edges, we have attributes), but it does not cover the category of hypergraphs, because hyperedges do not fit in this definition. In addition, we will ask the categories of graphs to have an initial object (the empty graph) and colimits that satisfy a given finitary property since it is needed in the completeness proof. The following definition characterizes the categories $\underline{\texttt{Graph}}$ satisfying those requirements.

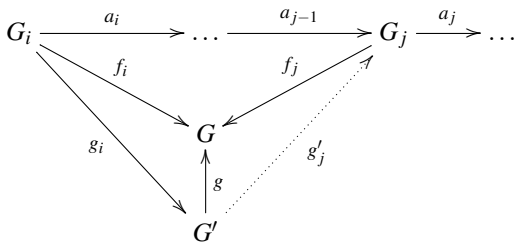
Definition 1 (Categories of Graphs). Consider a set of labels Σ. Let $\underline{\texttt{Graph}}$ be a category of graphs satisfying the following requirements:

1. Every $G \in \mathit{Objects}(\underline{\texttt{Graph}})$ is equipped (at least) with:
   - a set of nodes Node_G,
   - a set of edges Edge_G,

- source and target functions $s_G : Edge_G \to Node_G$ and $t_G : Edge_G \to Node_G$, respectively, and
- an edge labelling function $Label_G : Edge_G \to \Sigma$.

Moreover, we say that G is *finite* if $Node_G$ and $Edge_G$ are finite.

2. Graph morphisms $f : G_1 \to G_2$ in $\underline{\texttt{Graph}}$ consist (at least) of two functions $f_N : Node_{G_1} \to Node_{G_2}$, $f_E : Edge_{G_1} \to Edge_{G_2}$ such that
   - $f_N \circ s_{G_1} = s_{G_2} \circ f_E$, and $f_N \circ t_{G_1} = t_{G_1} \circ f_E$,
   - $Label_{G_2} \circ f_E = Label_{G_1}$
3. $\underline{\texttt{Graph}}$ is cocomplete and has an initial object. Moreover, colimits satisfy the following property called *FinCol*:
   Let $\{G_i \xrightarrow{a_i} G_{i+1}\}_{i \in \mathbb{N}}$ be an infinite sequence of morphisms between finite graphs and let $\{G_i \xrightarrow{f_i} G\}_{i \in \mathbb{N}}$ in $\underline{\texttt{Graph}}$ be their colimit. For any i and any finite graph G', if there are two morphisms $g_i : G_i \to G'$ and $g : G' \to G$, satisfying $g \circ g_i = f_i$, then there is some $j \geq i$ and a morphism $g'_j : G' \to G_j$ such that $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$, where a_{ij} denotes the morphism $a_{j-1} \circ \cdots \circ a_i$.



Intuitively, the property *FinCol* introduced in the above definition, states that if G can be seen as the union of a sequence of finite graphs, then if some other finite graph G' can be embedded in G then it should also be possible to embed G' in some graph in the sequence. Therefore, from now on, we will write $\underline{\texttt{Graph}}$ to denote any category satisfying the requirements in Def. 1. In particular, this is the case of the specific category of edge-labelled graphs, $\underline{\texttt{ELGraph}}$, whose objects and morphisms are defined exactly as required in Def. 1:

Definition 2 (The Category of Edge-Labelled Graphs). Given a set of labels Σ, an *edge-labelled graph* G over Σ consists (just) of:

- a set of nodes $Node_G$,
- a set of edges $Edge_G$,
- source, target, and labelling functions $s_G : Edge_G \to Node_G$, $t_G : Edge_G \to Node_G$, and $Label_G : Edge_G \to \Sigma$.

Edge-labelled graph morphisms $f : G_1 \to G_2$ consist of two functions $f_N : Node_{G_1} \to Node_{G_2}$, $f_E : Edge_{G_1} \to Edge_{G_2}$ such that

- $f_N \circ s_{G_1} = s_{G_2} \circ f_E$, and $f_N \circ t_{G_1} = t_{G_1} \circ f_E$,

- $Label_{G_2} \circ f_E = Label_{G_1}$

Edge-labelled graphs and morphisms form a category ELGraph.

Proposition 1 ELGraph *is cocomplete, has an initial object* $\emptyset$*, and satisfies the property FinCol.*

PROOF SKETCH. *(See Appendix A.1 for detailed proof)* The empty graph is the initial object in this category and the colimit construction is standard. Given a diagram D, the colimit object is the disjoint union of all the graphs in D quotiented by the equivalence induced by the morphisms in D.

Finally, to prove the property *FinCol*, we have to show that there is a G_j such that for every element (node or edge) $x' \in G'$, there is a unique $x_j \in G_j$ such that $f_j(x_j) = g(x')$. Then, we just have to show that $g'_j : G' \to G_j$, defined $g'_j(x') = x_j$, for each $x' \in G'$, satisfies $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$. ■

It is important to notice that not all categories having the elements (nodes, edges, etc.) required by Def. 1 may fail to satisfy the properties required in that definition.

In the following, we introduce the notions of path expression, pattern and pattern morphism, in order to define the category of patterns but, first, we characterize the properties that a class of languages must satisfy to properly label paths in patterns:

Definition 3 (Path Labelling Languages). Given a set of labels Σ, $\mathcal{L}(\Sigma)$ is a *class of path labelling languages over* Σ, if it satisfies the following conditions:

1. If $L \in \mathcal{L}(\Sigma)$ then $\emptyset \subset L \subseteq \Sigma^+$.
2. If $L_1, L_2 \in \mathcal{L}(\Sigma)$, then $L_1 \cdot L_2 \in \mathcal{L}(\Sigma)$.
3. If $L \in \mathcal{L}(\Sigma)$ then $L = S_0 \cup (\bigcup_{x \in S_1} \{x \cdot s |\ s \in L^x\})$ for some finite sets of labels $S_0, S_1 \subseteq \Sigma$, such that,
   - $\{x\} \in \mathcal{L}(\Sigma)$ for every $x \in S_0$, and
   - $L^x \in \mathcal{L}(\Sigma)$ and $\{x\} \cdot L^x \in \mathcal{L}(\Sigma)$ for every $x \in S_1$.

The set $d_L = \{\{x\} |\ x \in S_0\} \cup \{\{x\} \cdot L^x |\ x \in S_1\} \subseteq \mathcal{L}(\Sigma)$ is called the *finite disjunctive decomposition of* L.

The first condition in the above definition states that a class of graph labelling languages should not include the empty language, nor any language including the empty sequence. The reason is that this simplifies path unfolding. The second condition states that the class is closed under concatenation. The reason is that concatenation is needed for having path composition. Finally, the third condition is required to define the unfolding of labelled paths (Def. 10). Having finite disjunctive decomposition of labels means that every labelled path can be (successively) unfolded as a finite union of two sets of paths: the first one consists of paths with just one edge labelled with some symbol $x \in S_0$, and the second one consists of paths starting with an edge labelled with some symbol $x \in S_1$ followed by some path labelled with the language L^x in $\mathcal{L}(\Sigma)$. For instance, the language $L = \{a, abbc, acc, c\}$ can be (uniquely) decomposed as $d_L = \{\{a\}, \{c\}, \{a\} \cdot \{bbc, cc\}\}$ with $S_0 = \{a, c\}$ and $S_1 = \{a\}$. The idea, as

we will see later, is that the labeled path $\bullet_1 \overset{L}{\Rightarrow} \bullet_2$ can be unfolded to generate the finite set of path expressions $\{\bullet_1 \xrightarrow{a} \bullet_2,\ \bullet_1 \xrightarrow{c} \bullet_2,\ \bullet_1 \xrightarrow{a} \bullet \overset{\{bbc,cc\}}{\Longrightarrow} \bullet_2\}$.

Notice that the sets S_0 and S_1 are not necessarily disjoint but the languages in d_L are all disjoint since S_0 and S_1 are sets and $L^x \neq \emptyset$ for every $x \in S_1$.

If Σ is finite, the class of languages included in Σ^+ has disjunctive decompositions. So has the class of languages defined by regular expressions that do not include the empty string in their semantics, even if Σ is infinite. However, an arbitrary class of languages including a language $L \subseteq \Sigma^*$ may not have finite disjunctive decompositions if, for instance, the set S consisting of all symbols x such that there is a string in L starting by x, is infinite.

Now, we can define our category of patterns $\underline{\texttt{Pattern}}$ over $\underline{\texttt{Graph}}$:

Definition 4 (Graph Patterns and Path Expressions). Let $\mathcal{L}(\Sigma)$ be a class of path labelling languages over Σ. A *graph pattern* is a pair, $P = (G_P, \Rightarrow_P)$, consisting of a graph $G_P \in Objects(\underline{\texttt{Graph}})$ equipped with a relation $\Rightarrow_P \subseteq Node_{G_P} \times \mathcal{L}(\Sigma) \times Node_{G_P}$ whose elements, $\langle n, L, n' \rangle$ are called *path expressions*. Moreover, we say that P is *finite* if G_P is finite.

Along the paper, we may also write, $n \overset{L}{\Rightarrow}_P n'$ (or just $n \overset{L}{\Rightarrow} n'$ if P is clear from the context) to denote $\langle n, L, n' \rangle \in \Rightarrow_P$. Moreover, from now on we will assume that there is a fixed set of labels Σ and a fixed class of path labelling languages $\mathcal{L}(\Sigma)$.

Each path expression $\langle n, L, n' \rangle$ specifies a set of possible paths between the nodes n and n'. Intuitively, a graph G contains a path specified by $\langle n, L, n' \rangle$, if there is a path consisting of edges $e_1, \ldots, e_k$ in G with labels $x_1, \ldots, x_k$, such that n is the source of e_1, the source of e_i is the target of e_{i-1} for $1 < i \leq k$, n' is the target of e_k and the string $x_1 \cdots x_k$ is in L. In this sense, a pattern is the description of some requirements that are satisfied by the graphs matching the pattern. In particular, a graph G matching a pattern P must include some nodes, edges and paths matching the nodes, edges and paths in P.

To define precisely what it means that G matches P, we will first define the notion of morphism between two patterns P_1 and P_2 (see Def. 6). Roughly, a pattern morphism $f : P_1 \to P_2$ is a graph morphism, $f : G_{P_1} \to G_{P_2}$ such that for every path expression $\langle n, L_1, n' \rangle$ in P_1 there must be some (*derived*) paths in P_2 matching that path expression. This means that the paths matching $\langle n, L_1, n' \rangle$ in P_1 may not actually be in $\Rightarrow_{P_2}$ but they may be obtained composing paths in $\Rightarrow_{P_2}$ and edges in G_{P_2}. In this sense, prior to the definition of pattern morphism, we will define the notion of pattern closure, where the closure, $\widehat{P}$, of P is the pattern having the same underlying graph as P, G_P, but including the path expressions of all the derived paths in P.

At this point we may notice that any graph G can be identified with the pattern $(G, \emptyset)$, with the empty set of path expressions (see Remark 1). So a morphism from P to $(G, \emptyset)$ can be considered a morphism from P to G or, in other words, a morphism matching P to G. In particular, this morphism will map nodes and edges in G_P to nodes and edges in G, such that for every path expression $\langle n, L, n' \rangle$ in P there is some derived paths in G matching it.

Definition 5 (Edge Relation, Closed Patterns). Given a graph G, its *edge relation* $\to_G \subseteq Node_G \times \mathcal{L}(\Sigma) \times Node_G$, is defined as follows: $\langle n, \{x\}, n' \rangle \in \to_G$ if there is an edge $e \in Edge_G$ such that $s_G(e) = n$, $t_G(e) = n'$ and $Label_G(e) = x$.

Then, a pattern $P = (G_P, \Rightarrow_P)$ is *closed* if the following conditions are satisfied:

1. $\rightarrow_{G_P} \subseteq \Rightarrow_P$
2. If $\langle n_1, L_1, n_2\rangle, \langle n_2, L_2, n_3\rangle \in \Rightarrow_P$, then $\langle n_1, L_1 \cdot L_2, n_3\rangle \in \Rightarrow_P$

The closure of any pattern P, denoted $\widehat{P}$, is the least closed pattern such that $G_P = G_{\widehat{P}}$ and $\Rightarrow_P \subseteq \Rightarrow_{\widehat{P}}$.

Notice that, if P is closed, $\Rightarrow_P$ includes the specification of every finite path in G_P.

Definition 6 (Morphisms of Graph Patterns). A pattern morphism $f : P_1 \rightarrow P_2$ is a graph morphism $f : G_{P_1} \rightarrow G_{P_2}$ such that, for each $\langle n, L_1, n'\rangle \in \Rightarrow_{P_1}$, there is a path expression $\langle f_N(n), L_2, f_N(n')\rangle \in \Rightarrow_{\widehat{P}_2}$, such that $\emptyset \subset L_2 \subseteq L_1$.

Example and Notation. In order to avoid the use of too many figures, we will write many examples as text. Nodes will be denoted by bullets, $\bullet$, that may be subindexed by the name of the node like $\bullet_{BCN}$, or by an identifier like a number, e.g., $\bullet_1$. Edges will be denoted by simple arrows, e.g. $\bullet_{BCN} \xrightarrow{IB} \bullet_{CDG}$ denotes that there is an edge, with label *IB* from node *BCN* to node *CDG*. Path expressions are denoted by labelled double arrows, like $\bullet_{BCN} \overset{IB^+}{\Rightarrow} \bullet_{CDG}$. Graphs and patterns are surrounded by square brackets. For instance, $[\bullet_1 \xrightarrow{a} \bullet_2 \overset{b^+}{\Leftarrow} \bullet_3]$ denotes a pattern with three nodes, one edge labelled a, and one path expression labelled by the language denoted by b^+. Finally, morphisms are denoted by long arrows. Moreover, unless there is any ambiguity, we assume that morphisms map nodes to nodes with the same identifier, and edges to edges with the same label. For example, $[\bullet_1 \xrightarrow{a} \bullet_2 \overset{b^+}{\Leftarrow} \bullet_3] \longrightarrow [\bullet_1 \xrightarrow{a} \bullet_2 \xleftarrow{b} \bullet_4 \xleftarrow{b} \bullet_3]$ denotes the morphism mapping nodes 1, 2, and 3 and the edge labelled a in the first pattern into the corresponding nodes and edge. Moreover, we know that this is a morphism, because in the closure of the second pattern, which is actually a graph, there is a path, consisting of the composition of the edges going from 3 to 4 and from 4 to 2, whose label is the set $\{bb\}$, which is included in the language defined by b^+.

Proposition 2 (Category of Graph Patterns) *Patterns and pattern morphisms form the category* Pattern*. Moreover, if* Graph *has initial objects, is cocomplete and satisfies the property FinCol, then so does* Pattern*.*

PROOF SKETCH. *(See Appendix A.2 for the detailed proof)* As in the case of graphs, the empty pattern is the initial object in Pattern. Then, if D is a diagram in Pattern and $C = \{f_P : G_P \rightarrow H\}_{P \in D}$ is the colimit of the corresponding diagram in Graph, then we just have to prove that $C_{\texttt{Pattern}} = \{f_P : P \rightarrow Q\}_{f_P \in C}$ is the colimit of D in Pattern, where $Q = (H, \Rightarrow_Q)$ and:

$$\Rightarrow_Q = \{\langle f_P(n), L, f_P(n')\rangle \mid f_P \in C \text{ and } \langle n, L, n'\rangle \in \Rightarrow_P\}$$

Finally, the property FinCol is a consequence of how colimits are constructed in Pattern and of the fact that Graph satisfies the property. ■

Remark 1 (Graphs and Patterns)

1. *As a consequence of the above definitions, we have that every pattern P is isomorphic to its closure $\widehat{P}$ in* Pattern*. The reason is that any two patterns denoting the same graph with the same (derived) paths will be isomorphic.*
2. *Since most constructions used are defined up to isomorphism, we will not distinguish between isomorphic objects in the main algebraic constructions. Technically, this means that we consider that patterns are closed whenever it is convenient.*
3. *Technically, a graph G is not a pattern, but the category* Graph *can be represented as the full subcategory of* Pattern *whose objects P are of the form $(G,\emptyset)$, where $G \in$* Graph*.*
4. *Even if it is an abuse of notation, given a graph G and a pattern P, we will write $f : P \to G$ to denote that there is a pattern morphism $f : P \to (G,\emptyset)$.*

3. Graph Properties Expressed in GNL

In Section 2.1 we presented some examples to provide some intuition on *Graph Navigational Logic* (*GNL*). Now, we introduce it formally, studying some of its properties. More precisely, in the first subsection we define its syntax and semantics, adapting the nested notation defined in [3], discussing some alternative definitions, and introducing some properties of the logic. Then, in subsection 3.2, we present some constructions and results that are used in connection with our tableau method studied in Sections 4 and 5.

3.1. Graph Properties as Nested Conditions

To understand Habel and Pennemann's nested notation [3], we have to take into account the following basic ideas:

- In general, nested conditions do not state properties of graphs, but of morphisms from a given pattern P to a graph. More precisely, conditions are defined over a given *context* P and their models are morphisms $h : P \to G$. Intuitively, this means that a condition over P does not state a property of a graph G but of possible extensions of a given morphism from P to G.

  For example, given the context $P = [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2]$, then the morphism $f : P \to G$ matching nodes 1 and 2 to nodes $1'$ and $2'$ in G, satisfies the condition $\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \overset{x}{\to} \bullet_3 \overset{x}{\to} \bullet_2], \texttt{true})$ if there exists a node $3'$ in G such that $1'$ is connected to $3'$ by an x-labelled edge and $3'$ is connected to $2'$ by another x-labelled edge.

- Conditions over the empty context can be considered to state conditions over graphs.

- Let us suppose that in the previous example, in addition, we want to state that this node $3'$ does not have a loop, then we would state the condition

$$\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \overset{x}{\to} \bullet_3 \overset{x}{\to} \bullet_2], \neg c)$$

  where c would be a condition over the context $[\bullet_1 \overset{x}{\to} \bullet_3 \overset{x}{\to} \bullet_2]$ stating that there exists a loop on node 3. That is, we use nesting to state properties of the previous extension.

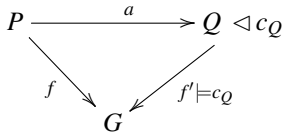
Then, the syntax and semantics of nested conditions are defined as follows:

Definition 7 (Conditions over Patterns, Nesting Level, Context, Satisfaction). A *condition* over a finite pattern P is defined inductively as follows:

- `true` is a condition over P. We say that `true` has nesting level 0.
- For every morphism $a : P \to Q$ and condition c_Q over a finite pattern Q with nesting level $n \geq 0$, $\exists(a, c_Q)$ is a condition over P with nesting level $n+1$.
- If c_P and c'_P are conditions over P with nesting level n and n', respectively, then $\neg c_P$ is a condition over P with nesting level n and $c_P \wedge c'_P$ is a condition over P with nesting level $max(n, n')$.
- As usual, `false` is an abbreviation for $\neg$`true`, $c_P \vee c'_P$ is an abbreviation for $\neg(\neg c_P \wedge \neg c'_P)$, and $\forall(a, c_Q)$ is an abbreviation for $\neg\exists(a, \neg c_Q)$.

Given a graph G (or, in general, any pattern[2] G), we inductively define when a morphism $f : P \to G$ *satisfies* a condition c_P over P, denoted $f \models c_P$:

- $f \models$ `true`.
- $f \models \exists(a, c_Q)$ if there exists $f' : Q \to G$ such that $f' \circ a = f$ and $f' \models c_Q$.
- $f \models \neg c_P$ if $f \not\models c_P$.



- $f \models c_P \wedge c'_P$ if $f \models c_P$ and $f \models c'_P$.

If c_P is a condition over P, we also say that P is the *context* of c_P.

[2] In Sections 5.2 and 5.3 , we will work with a special class of patterns as models of GNL.

Besides path expressions, there are some variations in the definitions of graph condition and satisfaction in the literature. In some approaches (e.g. [7]), morphisms in conditions are assumed to be monic, but in this paper (as in [3]) we assume that conditions may include arbitrary morphisms. Concerning satisfaction, in [3], Habel and Pennemann introduced two notions of satisfaction: *a-satisfaction* and *m-satisfaction*. The former notion is exactly what we have defined above. However, in *m*-satisfaction, the morphism f' in Def. 7 needs to be mono.

In addition, Habel and Pennemann showed that there are back and forth transformations that make both notions equivalent, in the sense that a morphism f m-satisfies a condition c if and only if f a-satisfies $t(c)$, for a given transformation t; and vice versa f a-satisfies c if and only if f m-satisfies $t'(c)$, for a given transformation t'. As a consequence of this equivalence, in general, most work of Habel and Pennemann has been done in terms of m-satisfaction, which they preferred. Moreover, since they also proved that every condition including arbitrary morphisms can be transformed into a condition including only monomorphisms having the same models (with respect to m-satisfaction), most often they also assume that conditions only include monos.

However, we have preferred to work with a-satisfaction because a-satisfaction provides some important practical advantages in the context of our work over m-satisfaction, as we will see in Remark 2.

Nested conditions are more general than needed, since they define properties on graph morphisms, rather than on graphs. As said above, graph properties are conditions over the empty pattern, since a morphism $\emptyset \to G$ can be considered equivalent to the graph G. However, we must notice that if $\exists(\emptyset \to P, c)$ is a graph property, then c is an arbitrary condition over P.

Definition 8 (*GNL* Syntax, *GNL* Semantics). The language of graph navigational properties, *GNL*, consists of all conditions over the empty pattern $\emptyset$. Conditions $\exists(\emptyset \to P, c_P)$ may also be denoted by $\exists(P, c_P)$. A graph (and, in general, a pattern) G satisfies a graph property c of *GNL*, denoted $G \models c$, if the unique morphism $e_G : \emptyset \to G$ satisfies c.

Example 1 (Properties on airport networks in *GNL*). The graph conditions, which are depicted in Fig. 3 using a semiformal notation, are formally written in *GNL* as follows, where $\Sigma = \{IB, AEA, AA\}$ and $L = \Sigma^+$:

1. $\exists(\emptyset \longrightarrow [\bullet_{BCN} \overset{IB}{\to} \bullet_{CDG} \overset{L}{\Rightarrow} \bullet_{LAX}],\ \texttt{true}) \vee \exists(\emptyset \longrightarrow [\bullet_{BCN} \overset{L}{\Rightarrow} \bullet_{JFK} \overset{AA}{\to} \bullet_{LAX}],\ \texttt{true})$
2. $\forall(\emptyset \longrightarrow [\bullet_1 \overset{L}{\Rightarrow} \bullet_2],\ \exists([\bullet_1 \overset{L}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \underset{L}{\overset{L}{\rightleftarrows}} \bullet_2],\ \texttt{true}))$
3. $\forall(\emptyset \longrightarrow [\bullet_1 \overset{IB^+}{\Rightarrow} \bullet_2],\ \exists([\bullet_1 \overset{IB^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \underset{IB^+}{\overset{IB^+}{\rightleftarrows}} \bullet_2],\ \texttt{true}))$
4. $\forall(\emptyset \longrightarrow [\bullet_1 \overset{IB}{\to} \bullet_2 \overset{L}{\Rightarrow} \bullet_3],\ \exists([\bullet_1 \overset{IB}{\to} \bullet_2 \overset{L}{\Rightarrow} \bullet_3] \longrightarrow [\bullet_1 \underset{IB}{\overset{IB}{\rightleftarrows}} \bullet_2 \underset{L}{\overset{L}{\rightleftarrows}} \bullet_3],\ \texttt{true}))$

As an example, let us explain the third condition above, to better understand its relation with the corresponding condition in Fig. 3.

This condition begins stating $\forall(\emptyset \longrightarrow [\bullet_1 \overset{IB^+}{\Rightarrow} \bullet_2], \dots)$ this is equivalent to say "For every morphism h matching the pattern $[\bullet_1 \overset{IB^+}{\Rightarrow} \bullet_2]$" to a graph G, i.e., "for any two

nodes n and n' in the given graph G, connected by a path formed by a sequence of edges labelled with *IB*".

Then, in the nested condition we have: $\exists([\bullet_1 \overset{IB^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \underset{IB^+}{\overset{IB^+}{\rightrightarrows}} \bullet_2], \texttt{true})$, that should be interpreted as "there should exist a morphism h' from the pattern $[\bullet_1 \underset{IB^+}{\overset{IB^+}{\rightrightarrows}} \bullet_2]$ to G that extends h, i.e., "there should exist a path from n' to n, formed by a sequence of edges labelled with *IB*".

We will now study some interesting aspects of GNL. First, we will show the existence of a *shift* transformation, allowing us to move conditions along morphisms. Then we will study some questions related to *split mono* morphisms when used in conditions.

Lemma 1 (Shift of Conditions over Morphisms) *For every pattern morphism $b : P \to P'$ and every condition c_p of context P, $\texttt{Shift}(b, c_P)$ is a condition of context P' defined inductively as follows:*

- $\texttt{Shift}(b, \texttt{true}) = \texttt{true}$.
- $\texttt{Shift}(b, \exists(a, c_Q)) = \exists(a', c_{Q'})$ *with* $c_{Q'} = \texttt{Shift}(b', c_Q)$ *such that (1) is a pushout.*

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

- $\texttt{Shift}(b, \neg c_P) = \neg \texttt{Shift}(b, c_P)$
- $\texttt{Shift}(b, c_P \wedge c'_P) = \texttt{Shift}(b, c_P) \wedge \texttt{Shift}(b, c'_P)$.

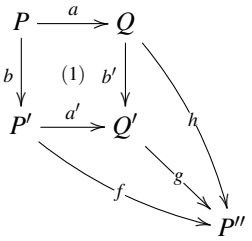
satisfying:

1. *The nesting level of* $\texttt{Shift}(b, c_P)$ *is not greater than the nesting level of* c_P.
2. *For each morphism* $f : P' \to P''$ *we have that* $f \models \texttt{Shift}(b, c_P) \Leftrightarrow f \circ b \models c_P$.

For generality, the property has been stated for arbitrary pattern morphisms, but it obviously applies to the special case when P' or P'' are graphs.

PROOF. The proof uses double induction on the structure and the nesting level of conditions. The base case is direct, since the nesting level does not change and every morphism satisfies $\texttt{true}$.

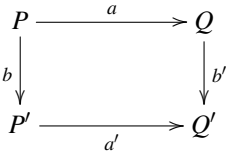
Suppose $c_P = \exists(a, c_Q)$, with nesting level $m+1$. If there is a morphism $f : P' \to P''$ such that $f \models \texttt{Shift}(b, \exists(a, c_Q))$, i.e., $f \models \exists(a', \texttt{Shift}(b', c_Q))$, according to diagram (1) below, then, there exists a morphism $g : Q' \to P''$ such that $g \models \texttt{Shift}(b', c_Q)$ and $f = g \circ a'$. Since (1) is a pushout, we have $f \circ b = g \circ a' \circ b = g \circ b' \circ a$ and, by inductive hypothesis, $g \circ b' \models c_Q$. Therefore, $f \circ b \models c_P$.



Conversely, if $f \circ b \models c_P$ there exists $h : Q \to P''$ such that $f \circ b = h \circ a$ and $h \models c_Q$. By the universal property of pushouts, there exists $g : Q' \to P''$ such that $f = g \circ a'$ and $h = g \circ b'$ and, by inductive hypothesis, $g \models \texttt{Shift}(b', c_Q)$. Hence, $f \models \texttt{Shift}(b, \exists(a, c_Q))$. In addition, $\exists(a', \texttt{Shift}(b', c_Q))$ has nesting level smaller or equal to $m+1$ since, again as a consequence of the inductive hypothesis $\texttt{Shift}(b', c_Q)$ has nesting level smaller or equal to m.

The rest of the cases easily follow from the inductive hypothesis and the definitions of satisfaction and nesting level. ■

Remark 2 *When working with m-satisfaction, the transformation needed to* `Shift` *a condition along a morphism, as in the above lemma, is different [3, 16]. More precisely, in that case,* $\texttt{Shift}(b, \exists(a, c_Q)) = \bigvee_{(a', b')} \exists(a', \texttt{Shift}(b', q))$*, such that* a', b' *are jointly epi and the diagram below commutes.*



We may notice that there are as many jointly epi a', b' *as different ways of overlapping* P' *and* Q*, which means that, in the worst case, the length of this disjunction is exponential on the size of* P' *and* Q*. Taking into account that* `Shift` *is the basis for two tableau rules (see Def. 12), working with our notion of satisfaction and the corresponding* `Shift` *transformation allows us to avoid this kind of exponential branching in our tableaux.*

The following property of `Shift` is used in some proofs in the paper:

Proposition 3 *Given morphisms* $g : P_1 \to P_2$ *and* $h : P_2 \to P_3$ *and given a condition* c_{P_1}*, we have:*

$$\texttt{Shift}(h, \texttt{Shift}(g, c_{P_1})) = \texttt{Shift}(h \circ g, c_{P_1})$$

PROOF SKETCH. *(See Appendix A.3 for the detailed proof)* The proof proceeds by induction on the structure of conditions. ■

If morphisms are assumed to be monic, as in [7], sets of negative conditions of the form $\neg\exists(a:P\to Q,\texttt{true})$ are always satisfiable by the identity, $id_P:P\to P$, provided that none of these morphisms a is an isomorphism. However, in GNL, sets of negative conditions may be unsatisfiable, even if a is not an isomorphism. One reason is that there are morphisms, called *split monos*, that are not isomorphisms but have a left inverse. For instance, the following example shows this kind of situation.

Example 2. Consider the negative condition $\neg\ell$, with:

$$\ell=\exists([\bullet_1\xrightarrow{x}\bullet_2]\ \xrightarrow{a}\ [\bullet_3\xleftarrow{x}\bullet_1\xrightarrow{x}\bullet_2],\texttt{true})$$

where a is the obvious inclusion. This condition is unsatisfiable because the identity morphism $id:[\bullet_1\xrightarrow{x}\bullet_2]\ \longrightarrow[\bullet_1\xrightarrow{x}\bullet_2]$ satisfies ℓ. The reason is that we can define a morphism $g:[\bullet_3\xleftarrow{x}\bullet_1\xrightarrow{x}\bullet_2]\ \longrightarrow[\bullet_1\xrightarrow{x}\bullet_2]$, where $g(\bullet_1)=\bullet_1$, $g(\bullet_2)=\bullet_2=g(\bullet_3)$ and where the two edges in the source pattern are mapped into the single edge in the target pattern, so that $g\circ a=id$.

Remark 3 (Split Monomorphisms)

1. *A morphism $a:P\to Q$ is* split mono *if there exists a morphism $a^{-1}:Q\to P$ such that $a^{-1}\circ a=id_P$. That is, if it has a left inverse.*
2. *If $a:P\to Q$ is split mono, $\exists(a:P\to Q,\texttt{true})$ is equivalent to* true*, i.e., it is a tautology: for any graph (or any pattern) G, and any morphism $f:P\to G$ it holds that $f\models\exists(a,\texttt{true})$ since there exists the morphism $g=f\circ a^{-1}:Q\to G$ satisfying $g\circ a=f\circ a^{-1}\circ a=f\circ id_P=f$, and $g\models\texttt{true}$.*
3. *If $a:P\to Q$ is split mono, but not an isomorphism, $\exists(a,c_Q)$ is not equivalent to c_Q. In fact, $\exists(a,c_Q)$ is a condition over P, while c_Q is a condition over Q.*
4. *We say that a set of conditions is* free of split monos*, if there are no split monos in their outer nesting level.*

The other situation where a set of negative conditions of the form $\neg\exists(a:P\to Q,c_Q)$ can be unsatisfiable, even if their morphisms are not split mono, is caused by the presence of path expressions in patterns and the implicit properties that they satisfy, as the following example shows.

Example 3. Consider the following two negative conditions:
$\ell_1=\neg\exists([\bullet_1\xRightarrow{x^+}\bullet_2]\ \xrightarrow{b_1}\ [\bullet_1\xrightarrow{x}\bullet_2],\texttt{true})$ and
$\ell_2=\neg\exists([\bullet_1\xRightarrow{x^+}\bullet_2]\ \xrightarrow{b_2}\ [\bullet_1\xrightarrow{x}\bullet\xRightarrow{x^+}\bullet_2],\texttt{true})$.
Morphisms b_1 and b_2 are not split mono, but the condition $c=\ell_1\wedge\ell_2$ is obviously unsatisfiable because, for every graph G, if there is a path from node 1 to node 2 satisfying the path expression x^+, then either that path consists only of an edge labelled x or it consists of an edge labelled x followed by a sequence of edges labelled x.

However, a set of conditions over P of the form $\neg\exists(a:P\to Q,c_Q)$ is satisfiable if they are free of split monos and path expressions:

Proposition 4 *If $a: P \to Q$ is not split mono and $\Rightarrow_P = \emptyset$, then id_P satisfies $\neg\exists(a, c_Q)$, for any c_Q.*

PROOF SKETCH. *(See Appendix A.4 for the detailed proof)* Since $P = (G_P, \emptyset)$, it is enough to show that $id_P = id_{G_P} : P \to G_P$ satisfies $\neg\exists(a, c_Q)$, for any c_Q, when a is not split mono. ■

3.2. Nested Conjunctive Normal Form, Lift, and Unfolding

In this section we study constructions that are used in our tableau approach, in Sections 4 and 5. First we present what is a condition in *nested conjunctive normal form* (*NCNF*). The reason is that our tableau method, presented in Sect. 4, assumes that conditions are in *NCNF*. Then, we present *lift transformations* that allow us to transform two conditions into another one. Finally, we present the concept of *unfolding* that, given a pattern P, allows us to state how to unfold a path expression in P.

Definition 9 (Literals, *NCNF*-conditions). A *positive* (resp. *negative*) *literal* ℓ is either `true` (resp. `false`) or a condition of the form $\exists(a,d)$ (resp. $\neg\exists(a,d)$).

A *clause* is a disjunction of literals.

A condition c is in *nested conjunctive normal form (NCNF)* if it is either `true`, or `false`, or a conjunction of clauses $c = \wedge_{j\in J} c_j$, with $c_j = \vee_{k\in K_j} \ell_{jk}$, where for each literal $\ell_{jk} = \exists(a_{jk}, d_{jk})$ or $\ell_{jk} = \neg\exists(a_{jk}, d_{jk})$, a_{jk} is not an isomorphism and d_{jk} is a condition in *NCNF*.

As usual, we will identify the condition in *NCNF* $c = c_1 \wedge c_2 \wedge \cdots \wedge c_k$ with the set of clauses $\{c_1, c_2, \ldots, c_k\}$. Moreover, we will say that ℓ is a literal in c if there is some $c_i \in c$, such that $c_i = \ell \vee c_i'$ for some (possibly empty) clause c_i'.

Any condition c_P can be transformed into an equivalent condition $|c_P|$ in *NCNF*. This transformation is standard, making use of the properties of boolean connectives (e.g., distributivity, associativity, de Morgan laws, etc), except for the elimination of isomorphisms which is based on the following equivalence:

$$\exists(a: P \to Q, c_Q) \equiv a(c_Q)$$

if a is an isomorphism, where $a(c_Q)$ denotes the following transformation:

- $a(\texttt{true}) = \texttt{true}$
- $a(\exists(b: Q \to Q', c_{Q'})) = \exists(b \circ a: P \to Q', c_{Q'})$
- $a(\neg c_Q) = \neg a(c_Q)$
- $a(c_Q \wedge c_Q') = a(c_Q) \wedge a(c_Q')$

Moreover, if c_P has nesting level n then $|c_P|$ has nesting level not greater than n. More precisely the transformations associated to boolean connectives do not modify the nesting level of a condition, but isomorphism elimination may reduce it. The proof that *NCNF* transformation preserves logical equivalence can be found in [4].

As a direct consequence of the Shifting Lemma 1, we can define a transformation `Lift` that, given two literals $\ell_1 = \exists(a_1, c_1)$ and ℓ_2, we obtain a new literal $\ell_3 = \texttt{Lift}(\ell_1, \ell_2)$ (pushing ℓ_2 inside ℓ_1) that is equivalent to the conjunction of ℓ_1 and ℓ_2:

Lemma 2 (Lift of Literals) *Let $\ell_1 = \exists(a_1, c_1)$ and ℓ_2 be literals with morphisms a_i: $P \to Q_i$, for $i = 1,2$. We define the lift of literals as follows:*

$$\texttt{Lift}(\exists(a_1, c_1), \ell_2) = \exists(a_1, c_1 \wedge |\texttt{Shift}(a_1, \ell_2)|)$$

Then, for every pattern morphism f, $f \models \ell_1 \wedge \ell_2$ if, and only if, $f \models \texttt{Lift}(\ell_1, \ell_2)$.
(See Appendix A.5 for the detailed proof).

For example, if:

$\ell_1 = \exists([\bullet_1] \xrightarrow{a_1} [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2], \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{a_2} [\bullet_1 \underset{x^+}{\overset{x^+}{\rightleftarrows}} \bullet_2], \texttt{true}))$

$\ell_2 = \exists([\bullet_1] \xrightarrow{b} [\bullet_1 \xrightarrow{y} \bullet_3], \texttt{true})$

then $\texttt{Lift}(\ell_1, \ell_2)$ is equal to $\exists([\bullet_1] \xrightarrow{a_1} [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2], c)$, where $c =$

$\Big(\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{a_2} [\bullet_1 \underset{x^+}{\overset{x^+}{\rightleftarrows}} \bullet_2], \texttt{true}) \wedge \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_3 \xleftarrow{y} \bullet_1 \overset{x^+}{\Rightarrow} \bullet_2], \texttt{true})\Big)$

Given a positive literal, ℓ_1, and a negative literal, ℓ_2, we can define another transformation, *partial lift*, that can be seen as a general version of resolution. In this case, the literal resulting from the lifting is just a consequence of the conjunction of ℓ_1 and ℓ_2.

Lemma 3 (Partial Lift of Literals) *Let $\ell_1 = \exists(a_1 : P \to Q_1, c_1)$ and $\ell_2 = \neg\exists(a_2 : P \to Q_2, c_2)$ such that there exists a morphism $g : Q_2 \to Q_1$ satisfying $a_1 = g \circ a_2$. We define the partial lift of literals as follows:*

$$\texttt{PLift}(\exists(a_1, c_1), \ell_2) = \exists(a_1, c_1 \wedge |\texttt{Shift}(g, \neg c_2)|)$$

Then, for every pattern morphism f, $f \models \ell_1 \wedge \ell_2$ implies $f \models \texttt{PLift}(\ell_1, \ell_2)$.
(See Appendix A.6 for the detailed proof).

For example, given the conditions:

$\ell_1 = \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_2} [\bullet_1 \xrightarrow{x} \bullet_2 \overset{x^+}{\Rightarrow} \bullet_3], \texttt{true})$, $\ell_2 = \neg\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_1} [\bullet_1 \xrightarrow{x} \bullet_2], \texttt{true})$

if the second condition holds, the first condition can not hold. Then, if g is the obvious inclusion of $[\bullet_1 \xrightarrow{x} \bullet_2]$ in $[\bullet_1 \xrightarrow{x} \bullet_2 \overset{x^+}{\Rightarrow} \bullet_3]$, the partial lift of ℓ_1 and ℓ_2 would be:

$$\texttt{PLift}(\ell_1, \ell_2) = \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_2} [\bullet_1 \xrightarrow{x} \bullet_2 \overset{x^+}{\Rightarrow} \bullet_3], \texttt{true} \wedge \neg\texttt{true}) \equiv \texttt{false}$$

But in general if

$$\ell_1 = \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_2} [\bullet_1 \xrightarrow{x} \bullet_2 \overset{x^+}{\Rightarrow} \bullet_3], c_1), \quad \ell_2 = \neg\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_1} [\bullet_1 \xrightarrow{x} \bullet_2], c_2)$$

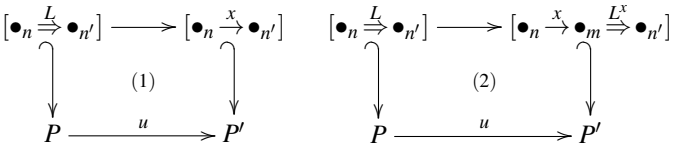
then $\texttt{PLift}(\ell_1, \ell_2) = \exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_2} [\bullet_1 \xrightarrow{x} \bullet_2 \overset{x^+}{\Rightarrow} \bullet_3], c_1 \wedge |\texttt{Shift}(g, \neg c_2)|)$

Given a pattern P including a path expression, the third kind of operation allows us to define a condition, actually a tautology, that states the forms in which we can unfold a path expression in P.

Definition 10 (Unfolding Morphisms). Consider a pattern P, a path expression $\langle n, L, n' \rangle \in \Rightarrow_P$, such that $d_L = \{\{x\} \mid x \in S_0\} \cup \{\{x\} \cdot L^x \mid x \in S_1\}$ is the finite disjunctive decomposition of L (c.f. Def. 3). We define the set of *unfolding morphisms* of P with respect to $\langle n, L, n' \rangle$, written $\texttt{Unfold}(P, \langle n, L, n' \rangle)$, as:

$\{u : P \to P' \mid \{x\} \in d_L$ and u is defined by pushout (1)$\} \cup$

$\{u : P \to P' \mid \{x\} \cdot L^x \in d_L$ and u is defined by pushout (2)$\}$



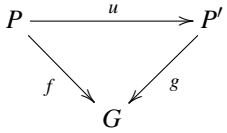
If the context is clear, we may also say that u in the above definition is the unfolding associated to the language $\{x\}$, respectively to the language $\{x\} \cdot L^x$.

For instance, according to Def. 3, the language L denoted by the regular expression $a \cup ab^+ \cup ac^+ \cup c$ has the finite disjunctive decomposition $d_L = \{\{a\}, \{c\}, \{a\} \cdot L^a\}$, with $S_0 = \{a, c\}$, $S_1 = \{a\}$, and L^a being the language denoted by the regular expression $b^+ \cup c^+$. So, according to the above definition, if, for instance, $P = [\bullet_n \overset{L}{\Rightarrow} \bullet_{n'}]$ then $\texttt{Unfold}(P, \langle n, L, n' \rangle) = \{u_0 : [\bullet_n \overset{L}{\Rightarrow} \bullet_{n'}] \longrightarrow [\bullet_n \overset{a}{\to} \bullet_{n'}], \quad u_1 : [\bullet_n \overset{L}{\Rightarrow} \bullet_{n'}] \longrightarrow [\bullet_n \overset{c}{\to} \bullet_{n'}], \quad u_2 : [\bullet_n \overset{L}{\Rightarrow} \bullet_{n'}] \longrightarrow [\bullet_n \overset{a}{\to} \bullet_m \overset{L^a}{\Longrightarrow} \bullet_{n'}]\}$.

Path unfolding can be characterized by the following proposition:

Proposition 5 (Unfolding Tautologies) *Given a pattern P and a path expression $\langle n, L, n' \rangle \in \Rightarrow_P$, we have that the condition $\bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$ is a tautology over P.*

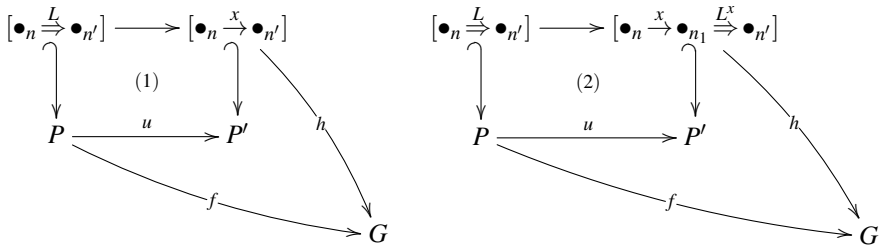
PROOF. We have to prove that $f \models \bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$ for every $f : P \to G$. That is, for every f there exists $u : P \to P' \in \texttt{Unfold}(P, \langle n, L, n' \rangle)$ and $g : P' \to G$ such that $g \circ u = f$.



Since f is a pattern morphism and G is a graph, if $f(n) = m$ and $f(n') = m'$ there must be a $\langle m, \{s\}, m' \rangle \in \to_G^+$, such that $s \in L = \bigcup_{x \in S_0} \{x\} \cup \bigcup_{x \in S_1} \{x\} \cdot L^x$. This means that $s = x$ for some $x \in S_0$, or $s = x \cdot s'$ for some $x \in S_1$ and $s' \in L^x$. Then, we can define a morphism h for each case:

- $h : [\bullet_n \overset{x}{\to} \bullet_{n'}] \longrightarrow G$, with $h(n) = m, h(n') = m'$, and $h([\bullet_n \overset{x}{\to} \bullet_{n'}]) = [\bullet_m \overset{x}{\to} \bullet_{m'}]$ since $\langle m, \{x\}, m' \rangle \in \to_G$
- $h : [\bullet_n \overset{x}{\to} \bullet_{n_1} \overset{L^x}{\Rightarrow} \bullet_{n'}] \longrightarrow G$, with $h(n) = m, h(n_1) = m_1, h(n') = m'$, and $h([\bullet_n \overset{x}{\to} \bullet_{n_1}]) = [\bullet_m \overset{x}{\to} \bullet_{m_1}]$ since $\langle m, \{x\}, m_1 \rangle \in \to_G$ and $\langle m_1, \{s'\}, m' \rangle \in \to_G^+$

Hence, by definition of h, we have the following commuting diagrams, where u is the unfolding morphism associated to the language $\{x\}$ or $\{x\}\cdot L^x$, respectively:



and by the universal property of pushouts, there is $g : P' \to G$ such that $g \circ u = f$. ■

4. Tableau Reasoning for GNL

Tableaux are a standard refutation technique for theorem proving that is used in the context of many logics (see, e.g. [17]). A tableau is a tree that represents the set of formulas that we want to refute. A branch in a tableau is the representation of the conjunction of formulas in the branch, and a tableau represents the disjunction of all the formulas represented by its branches. Tableaux are constructed by some given rules. These rules allow us to place in the tableau new formulas obtained by some form of decomposition of the given ones. The goal is to generate enough of these formulas to find contradictions among them. In our case, we use the standard tableau rule from Propositional Logic to decompose conjunctions and disjunctions but, in addition, we use some inference rules whose results are also placed in the tableau, with the same aim of generating contradictions. When we detect a contradiction in a branch of a tableau, we *close* the branch. If at some point all the branches of the tableau are closed, we consider that the given set of formulas has been refuted. Conversely, if there are open branches and we have not postponed indefinitely the application of some inferences, we consider that the given set of formulas is satisfiable. Obviously, if satisfiability is undecidable, the construction of a tableau may never end. However, soundness and completeness ensure that a given set of formulas is unsatisfiable if and only if an associated tableau is closed in finite time.

In our case, the nested structure of conditions makes it very difficult to check satisfiability using standard tableaux. For this reason, we use the notion of nested tableaux [7] that fits adequately in our framework, where the elements of a nested tableau are standard tableaux that we call *basic tableaux*. More precisely, in the first subsection of this section, we introduce basic tableaux, together with the tableau rules to build them, and in the second subsection, we study our notion of nested tableaux.

4.1. Basic Tableaux for GNL

As usual, the formulas in our tableaux are literals. The construction of a tableau for a condition c_P in *NCNF* is roughly as follows. We start with a tableau consisting of the single node `true` and for every clause $\ell_1 \vee \cdots \vee \ell_n$ in c_P we extend all the leaves in the

tableau with n branches, one for each literal ℓ_i. In addition to the standard *extension* rule (ext), the rules that are specific for GNL are the *lift* rule (L), based on Lemma 2 (following [18, 4]); the *partial lift* rule (PL), based on Lemma 3; the unfolding rule (U), based on the construction described in Def. 10 and on Prop. 5; and the *split reduction* rule (SR), to be applied on negative literals $\neg\exists(a,c)$ where a is split mono, based on Rem. 3. The literals that we add to a given branch, in the case of the extension rule, are logical consequences of the given condition c_P, while in the case of the lift and partial lift rules, are logical consequences of literals in that branch. Finally, in the case of the latter two rules (unfolding and split reduction), they are tautologies.

All rules are depicted in Fig. 4 and they are formally defined below in Def. 12.

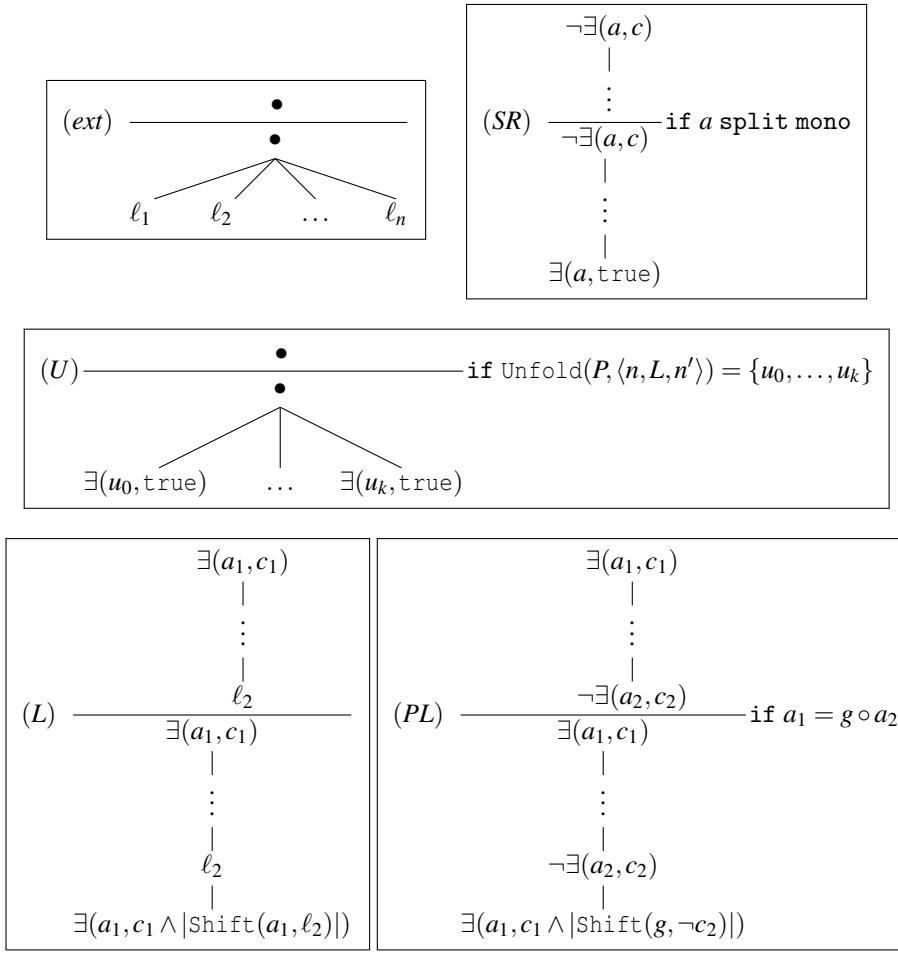


Figure 4: Tableau Rules

Definition 11 (BasicTableau and Branch of Context P). Given a finite pattern P, a

basic tableau of context P (or just a tableau) is a finitely branching tree whose nodes are literals over P in *NCNF*. A *branch* in a tableau T is a maximal path in T.

Definition 12 (Basic Tableau Rules). Given a condition $c_P = \{c_{1P}, c_{2P}, \ldots, c_{kP}\}$ in *NCNF*, a basic tableau of context P for c_P is constructed using the following rules:

- **Initial** *(init)*: A tree consisting of the single node $\texttt{true}$ is a tableau.
- **Extension rule** *(ext)*: If the clause $\ell_1 \vee \ldots \vee \ell_n$ is in c_P, we can extend any branch B with n descendants $\ell_1, \ldots \ell_n$.
- **Lift** *(L)*: If a given branch B includes the literals $\ell_1 = \exists(a_1, c_1)$ and ℓ_2, then we can extend B with the literal $\exists(a_1,\ c_1 \wedge |\texttt{Shift}(a_1, \ell_2)|)$.
- **Partial Lift** *(PL)*: If a given branch B includes the literals $\ell_1 = \exists(a_1 : P \to Q_1, c_1)$ and $\ell_2 = \neg\exists(a_2 : P \to Q_2, c_2)$ such that there exists a morphism $g : Q_2 \to Q_1$ satisfying $a_1 = g \circ a_2$, then we can extend B with the literal $\exists(a_1, c_1 \wedge |\texttt{Shift}(g, \neg c_2)|)$.
- **Split Reduction** *(SR)*: If a given branch B includes a negative literal $\neg\exists(a, c)$ with a split mono, then we can extend B with the literal $\exists(a, \texttt{true})$.
- **Unfolding** *(U)*: If $\langle n, L, n' \rangle \in \Rightarrow_P$, then we can extend any branch B with all the elements in the set $\{\exists(u, \texttt{true})\}_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)}$ as descendants.

Notice that, given a condition c_P, if $\langle n, L, n' \rangle \in \Rightarrow_P$, then using the unfolding rule (U) is equivalent to extending the original condition c_P with the tautology $\bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$ and, then, applying the rule (*ext*).

Definition 13 (Open/Closed Branch and Tableau). In a tableau T, a branch B is *closed* if B contains $\exists(a, \texttt{false})$ or $\texttt{false}$; otherwise, it is *open*. A tableau is *closed* if all its branches are closed.

Example 4. Consider the graph G of flights given in Fig. 1. This graph can be represented by the condition $c = \exists(\emptyset \longrightarrow P, true)$, where $P = (G, \emptyset)$. Assume now a property (or boolean query) expressing that there exists a direct IB flight from BCN to CDG and a connection from CDG to LAX operated by a sequence of one or more IB flights, followed by an AA flight. This property is expressed by the condition $c_1 = \exists(\emptyset \longrightarrow P_1, true)$, where $P_1 = [\bullet_{BCN} \overset{IB}{\to} \bullet_{CDG} \overset{IB^+ \cdot AA}{\Longrightarrow} \bullet_{LAX}]$. More precisely, $P_1 = (G_1, \Rightarrow_{P_1})$, with the graph G_1 consisting of nodes BCN, CDG, and LAX, and one IB labelled edge from BCN to CDG, together with the path expresion $\langle CDG, IB^+ \cdot AA, LAX \rangle$ in $\Rightarrow_{P_1}$. Then we can see that G satisfies this property by proving that $c \wedge \neg c_1$ is not satisfiable using the closed tableau for $c \wedge \neg c_1$, depicted in Fig. 5. The construction of this tableau can be explained as follows: after adding c and $\neg c_1$ by the Extension rule (ext), the Partial Lift rule (PL) can be applied to c and $\neg c_1$ since there exists a pattern morphism $g : P_1 \to P$. This pattern morphism is the inclusion morphism $g : G_1 \to G$, satisfying that, for the path expression $\langle n, L, m \rangle = \langle CDG, IB^+ \cdot AA, LAX \rangle \in \Rightarrow_{P_1}$, there exists $L' = \{IB \cdot AA\} \subseteq L$ such that $\langle g(n), L', g(m) \rangle = \langle CDG, \{IB \cdot AA\}, LAX \rangle$ belongs to $\Rightarrow_{\widehat{P}} = \to_G^+$.

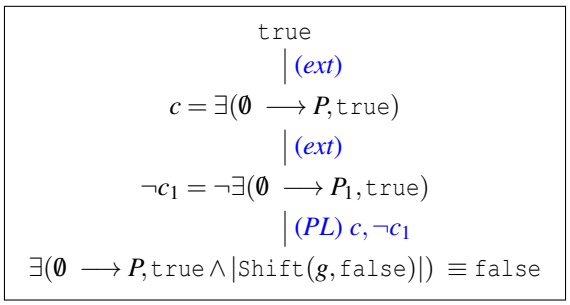


Figure 5: Closed tableau for proving a property on the airport graph

Example 5. Consider the condition $c_P = \ell_1 \wedge \ell_2$ with literals as in Example 3
$\ell_1 = \neg\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_1} [\bullet_1 \xrightarrow{x} \bullet_2], \texttt{true})$ and
$\ell_2 = \neg\exists([\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \xrightarrow{b_2} [\bullet_1 \xrightarrow{x} \bullet \overset{x^+}{\Rightarrow} \bullet_2], \texttt{true})$
and whose context P is the pattern $[\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2]$. Then, in Fig. 6 we show a closed tableau for c_P.

Notice that, according to Def. 3, the language [3] $L = x^+$ has the finite disjunctive decomposition $d_L = \{\{x\}, \{x\}\cdot x^+\}$ so, according to Def. 10, $\texttt{Unfold}(P, \langle \bullet_1, x^+, \bullet_2 \rangle) =$

$$\{u_0 : [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \xrightarrow{x} \bullet_2], \quad u_1 : [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2] \longrightarrow [\bullet_1 \xrightarrow{x} \bullet \overset{x^+}{\Rightarrow} \bullet_2]\}$$

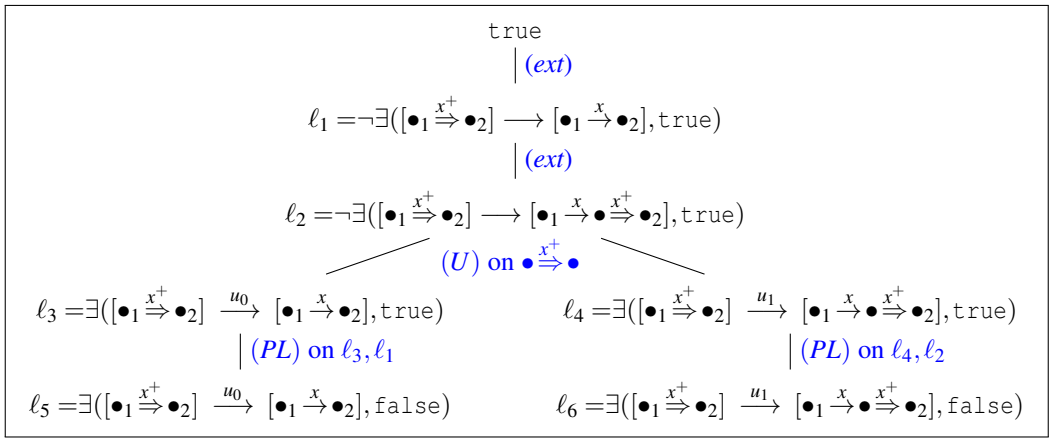


Figure 6: Closed tableau

The tableau rules may generate contradictions at the outer level of nesting for the literals in a given condition, as seen in the examples in Fig. 5 and Fig. 6, but this is not

[3] In an abuse of notation we may write, for instance, x^+ for the language L denoted by the regular expression x^+.

enough, because contradictions may occur at inner levels of nesting. Instead of defining additional rules that would do something similar at any nesting level, what we do is to associate additional tableaux for each level. Our procedure can be roughly described as follows. After applying the extension rule *(ext)* to all clauses in the given condition, if a branch has only negative literals free of split monos, and there are no path expressions in the context P, then we can stop the construction, since Prop. 4 guarantees that this branch (and, hence, the given condition) has a model. Otherwise, we may apply the split reduction rule (SR), if there is some negative literal including a split mono, and a finite number of times the unfolding rule (U), if there are some path expressions included in the context. Notice that, at this point, we would have some positive literals in the branch: the positive literals that were in the branch at the beginning, if there were any, plus the literals added by the split reduction and the unfolding rules. Then, we have that, for every branch B, either we are able to close it, or we can choose a given positive literal and use the lifting rule (L) and the partial lifting rule (PL), as many times as needed, until we produce a literal which is equivalent to the conjunction of literals in B (see Lemmas 2 and 3).

For instance, if $\ell_1, \ldots, \ell_n$ are the literals in the branch, it is enough to choose a positive literal, say $\ell_1 = \exists(a_1 : P \to Q_1, c_1)$, that we call the *hook* of the branch, and to successively apply the lift rule and the partial lift rule (if possible), first to ℓ_1 and ℓ_2, next to the result and ℓ_3 and so on, until we have applied the rules to all the literals in the branch.

To ensure that our proof method finds a refutation if the given property is unsatisfiable, we must guarantee that we have applied all needed inferences, which means that the procedure is fair, in the sense that no inference is postponed infinitely. When working with tableaux, fairness is guaranteed by the notion of saturation. In our case, since we work at two levels, we use two notions of saturation, one for basic tableaux, that we call *semi-saturation* and the other for nested tableaux (see Def. 22). We use the term semi-saturation because we do not require that all our basic tableaux include the results of applying all possible tableaux rules. For instance, if a branch B includes positive literals, we only require that the lift rule has been applied in such a way that the leaf of B includes, in its inner condition, all the literals in the branch (properly shifted), except the hook, plus the result of all possible applications of the partial lift rule to the hook and to any negative literal. For similar reasons, we do not require that the unfolding rule has been applied to all path expressions of the given context. By the way, notice that if in the context there are n path expressions, then we can apply the unfolding rules at most n times. Finally, we only require that a branch includes the application of a Split Reduction rule if the rest of the literals are negative, because it is not needed otherwise.

Definition 14 (Semi-saturation, Hook for a Branch). Given a *tableau* T for a condition c_P over P, we say that T is *semi-saturated* if no new literals can be added to any branch in T using the extension rule, and for every branch B in T one of the following conditions hold:

1. B is closed.
2. All the literals in B are negative, free of split monos and $\Rightarrow_P = \emptyset$.

3. There is a positive literal $\ell = \exists(a : P \to Q, c)$ in B, called the *hook* for the branch B in T, such that the literal in the leaf of B is

$$\ell_{leaf} = \exists(a,\ c \wedge \bigwedge_{\ell' \in B \setminus \{\ell\}} |\texttt{Shift}(a, \ell')| \wedge \bigwedge_{\ell' \in B^{PL}} |\texttt{Shift}(g, \neg d')|)$$

where $B^{PL} = \{\neg\exists(a', d') \in B \setminus \{\ell\} \mid \exists g : a = g \circ a'\}$.

Example 6 (Semi-saturated Tableau). Let ℓ_1 and ℓ_2 be the literals in Example 3 and let $c_\emptyset$ be a condition over the empty context consisting of the following conjunction of literals: $\exists(\emptyset \xrightarrow{a_1} [\bullet \overset{x^+}{\Rightarrow} \bullet], \ell_1 \wedge \ell_2) \wedge \exists(\emptyset \xrightarrow{a_2} [\bullet], \ell_3)$, where $\ell_3 = \exists([\bullet_1] \xrightarrow{b_3} [\bullet_1 \overset{x^+}{\Rightarrow} \bullet_2], \texttt{true})$.

Then, the tableau in Fig. 7 is the result of, first, applying the extension rule on $c_\emptyset$, to obtain c_1 and c_2, and, then, the lift rule, where the literal $c_1 = \exists(a_1, \ell_1 \wedge \ell_2)$ has been chosen as hook. The tableau is open but semi-saturated as it satisfies Def. 14.

We may notice that we have been unable to prove that c_1 is unsatisfiable, even if from Example 5 we know that $\ell_1 \wedge \ell_2$ is unsatifiable, which means that c_1 is also unsatisfiable. As we will see in the following section, this refutation can be completed using a nested tableau.

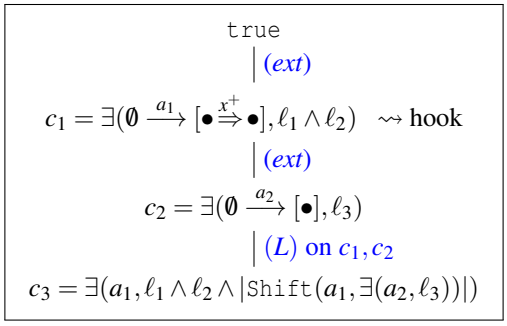


Figure 7: Semi-saturated tableau of context $\emptyset$

Following the procedure described above, we can build a finite semi-saturated tableau for any condition in *NCNF* that includes the application of any finite number of unfolding rules. This is needed because for completeness we will need to ensure that the unfolding rule has been applied enough times to find possible contradictions.

Lemma 4 (Existence of Finite Semi-saturated Tableaux) *For any conditon c_P over P in NCNF and any natural number k, such that k is smaller or equal than the number of path expressions in $\Rightarrow_P$, there exists a finite semi-saturated tableau T for c_P including k unfolding tautologies.*

PROOF SKETCH. *(See Appendix A.7 for the detailed proof)* To obtain a semisaturated tableau it is enough to apply the procedure described above, immediately before Def.

14. Moreover, since we apply a finite number of rules, the resulting tableau will be finite. ■

To end this section, we show the soundness of our tableau method.

Definition 15 (Branch and Tableau Satisfiability). A tableau T is *satisfiable* if it includes a branch B that is satisfiable, meaning that there is a morphism $f : P \to G$ satisfying all the literals in B. In this case, we say that $f : P \to G$ is a model for B, and also for T, and we write $f \models B$ and $f \models T$.

Even if the soundness proof is relatively simple, we will first show, as a first lemma, that the given tableau rules are sound, in the sense that they preserve satisfiability, and then we will show that if the given tableau rules are sound then the tableau method is sound. The reason is that, structuring the proof in this manner will allow us to show, in a very simple way, soundness of the same tableau method for a different class of models (Theorem 2).

Definition 16 (Soundness of Tableau Rules). A tableau rule that, given a branch B, extends B with literals $\ell_1, \ldots, \ell_n$, *is sound* if $f \models c_P$ and $f \models B$ imply $f \models \ell_1 \vee \cdots \vee \ell_n$, for any c_P in *NCNF*.

Lemma 5 (Soundness of rules) *The tableau rules given in Def. 12 are sound.*

PROOF. We consider each rule:

- (init): For every f, $f \models \texttt{true}$.
- (ext) If $\ell_1 \vee \cdots \vee \ell_n$ is a clause in c_P, obviously, if $f \models c_P$ then $f \models \ell_1 \vee \cdots \vee \ell_n$.
- (L) If we apply the lift rule on literals ℓ_1 and ℓ_2 in B, by Lemma 2, we know that the resulting literal ℓ_3 is equivalent to $\ell_1 \wedge \ell_2$. Thus $f \models \ell_3$.
- (PL) If we apply the partial lift rule on literals ℓ_1 and ℓ_2 in B, by Lemma 3, we know that the resulting literal ℓ_3 is a consequence of $\ell_1 \wedge \ell_2$. Thus $f \models \ell_3$.
- (SR) If a is split mono, according to 2. in Remark 3, $\ell = \exists(a : P \to Q, \texttt{true})$, is a tautology. Hence, $f \models \ell$.
- (U) If we apply the unfolding rule, for some path expression $\langle n, L, n' \rangle \in \Rightarrow_P$, by Prop. 5, we know that $\bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$ is a tautology. Hence $f \models \bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$. ■

Lemma 6 (Soundness of tableaux) *Given a condition c_P in NCNF, if we build a basic tableau T for c_P using sound rules, then if c_P is satisfiable so is T.*

PROOF. Let $f : P \to G$ be a morphism such that $f \models c_P$, we show by induction on the construction of T that $f \models T$.

- If T consists only of the node $\texttt{true}$, $f \models T$.

- For the general case, suppose that $f \models T$. We have to show that if T' is constructed by applying any basic tableau rule to T, then $f \models T'$. By inductive hypothesis, we may assume that there exists a branch B in T such that $f \models B$. If this branch is not extended using some tableau rule when constructing T', then it directly holds that $f \models T'$, since B is still a branch in T'.
- Suppose that the rule applied to construct T' extends B with literals $\ell_1, \ldots, \ell_n$, we can show that there exists a branch B' in T' extending B such that $f \models B'$ and therefore $f \models T'$. In particular, since the rule is assumed to be sound, according to Def. 16, we know that $f \models \ell_1 \vee \cdots \vee \ell_n$ and, thus, there must be some i, with $1 \leq i \leq n$, such that $f \models \ell_i$. But this means that if B' is the extension of B including ℓ_i, we have that $f \models B'$. ■

Finally, the following lemma will be required for proving completeness.

Lemma 7 (Semi-saturation and Satisfiability) *If T is a semi-saturated tableau for a condition c_P in NCNF, then we have:*

1. *$f \models T$ implies $f \models c_P$.*
2. *For every $c'_P \subseteq c_P$ and every open branch B, if for every literal ℓ in B that is also in c'_P $f \models \ell$, then $f \models c'_P$.*
3. *If for every literal ℓ in B $f \models \ell$, then $f \models c_P$.*

(See Appendix A.8 for the detailed proof).

4.2. Nested Tableaux for GNL

As discussed above, basic tableaux cannot be used in a simple way as a proof procedure for properties in *NCNF*. We use a notion of *nested tableaux* whose idea is that, for each open branch of a tableau T whose literal in the leaf is $\exists(a:P \to Q, c_Q)$, we open a new tableau T' to try to refute condition c_Q. Then, we say that $\exists(a, c_Q)$ is the *opener* for T'.

Example 7 (Tableau from an Opener). By using the leaf of the open branch in the semi-saturated tableau in Fig. 7 as opener,

$$c_3 = \exists(a_1 : \emptyset \xrightarrow{a_1} [\bullet \overset{x^+}{\Rightarrow} \bullet], \ell_1 \wedge \ell_2 \wedge |\texttt{Shift}(a_1, \exists(a_2, \ell_3))|)$$

we can open a new tableau of context $\bullet \overset{x^+}{\Rightarrow} \bullet$, for the nested condition of c_3, $\ell_1 \wedge \ell_2 \wedge |\texttt{Shift}(a_1, \exists(a_2, \ell_3))|$, as depicted in Fig. 8. Then this new tableau can be closed, as shown in Fig. 6, since a contradiction arises from literals ℓ_1 and ℓ_2.

Nested tableaux have nested branches consisting of sequences of branches of a sequence of tableaux. While our basic tableaux are assumed to be finite, nested tableaux and nested branches may be infinite. For example, this will be the case when the given condition includes a path expression like $\bullet_n \overset{a^+}{\Rightarrow} \bullet_{n'}$. In that case, to generate a contradiction, we may need to unfold this path expression consecutively an unbounded number of times, that is:

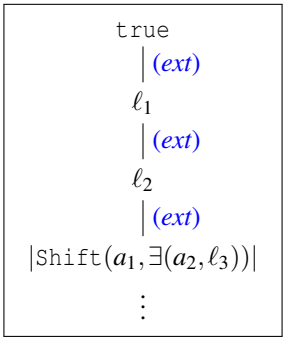


Figure 8: Tableau of context $\bullet \overset{x^+}{\Rightarrow} \bullet$

$$\bullet_n \xrightarrow{a} \bullet_1 \overset{a^+}{\Rightarrow} \bullet_{n'}, \quad \bullet_n \xrightarrow{a} \bullet_1 \xrightarrow{a} \bullet_2 \overset{a^+}{\Rightarrow} \bullet_{n'}, \quad \bullet_n \xrightarrow{a} \bullet_1 \xrightarrow{a} \bullet_2 \xrightarrow{a} \bullet_3 \overset{a^+}{\Rightarrow} \bullet_{n'}, \quad \ldots$$

So, if we do the first unfolding at a certain tableau t, we would only be able to apply the second unfolding at the next tableau, and so on. Since the number of unfoldings needed is unbounded, in the limit, we will have to unfold that path expression infinitely many times, which means that the nested tableau will be infinite. Nevertheless, we must point out that we may also have infinite nested tableaux if the given conditions do not include path expressions [7].

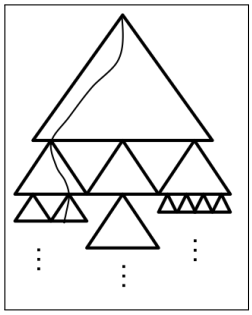


Figure 9: Nested tableau with nested branch

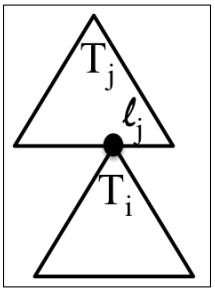


Figure 10: Opener ℓ_j

Definition 17 (Nested Tableau, Opener, Nested Branch, Semi-Saturation). Consider that $(I, \leq, i_1)$ is a poset with a least element i_1. A *nested tableau NT* for a condition c is $(T_{i_1}, \{t_i\}_{i \in I \setminus \{i_1\}})$ where:

1. T_{i_1} is a tableau for c with empty context, called the *initial tableau*.
2. For every $i \in I \setminus \{i_1\}$, $t_i = \langle T_i, j, B_j, \ell_j \rangle$ where
   - B_j is a branch in some tableau T_j, whose leaf is $\ell_j = \exists(a_j : P_j \to P_i, c_i)$, with $j \in I$ and $j < i$,
   - T_i is a tableau for c_i with context P_i and ℓ_j is called its *opener*.

If T_i is the initial tableau in NT (that is, $i = i_1$) or if there is a 4-tuple $t_i = \langle T_i, j, B_j, \ell_j \rangle$ in NT then we will say that T_i is a tableau in NT.

A *nested branch NB* in NT 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, then $\langle T_{i_{k+1}},i_k,B_{i_k},\ell_{i_k}\rangle$ is in NT.

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

We may notice, that when we have two consecutive tableaux, T_j and T_i, that is, when $\langle T_i,j,B_j,\exists(a_j:P_j\to P_i,c_i)\rangle\in NT$, we have implicitly defined a morphism between their contexts. More precisely, the context of T_j is P_j and the context of T_i is P_i, then the morphism associated to these consecutive tableaux is $a_j:P_j\to P_i$. Similarly, we have an implicit sequence of morphisms defined for any nested branch $B_{i_1},B_{i_2},\ldots,B_{i_k},B_{i_{k+1}},\ldots$, that is, $P_{i_1}\xrightarrow{a_{i_1}}P_{i_2}\xrightarrow{a_{i_2}}P_{i_3}\xrightarrow{a_{i_3}}\ldots\xrightarrow{a_{i_{k-1}}}P_{i_k}\xrightarrow{a_{i_k}}P_{i_{k+1}}\xrightarrow{a_{i_{k+1}}}\ldots$, where $P_{i_1},P_{i_2}\ldots,P_{i_k},P_{i_{k+1}},\ldots$ are, respectively, the contexts of tableaux $T_{i_1},T_{i_2}\ldots,T_{i_k},T_{i_{k+1}},\ldots$.

Notation From now on, given a nested branch NB and its associated sequence of contexts, we will denote by a_{ij} the composition of morphisms between P_i and P_j, i.e., $a_{ij}:P_i\xrightarrow{a_i}\circ\ldots\circ\xrightarrow{a_{j-1}}P_j$. Moreover, since the context of any initial tableau T_{i_1} is the empty pattern, we may just write $\emptyset\to P_j$ to denote the morphism $a_{i_1j}:\emptyset\to P_j$.

Definition 18 (Nested Tableau Rules). Given a graph property c in *NCNF*, a *nested tableau* for c is constructed with the following rules:

- **Initialization (I)**: If c is a condition over $\emptyset$ and T_{i_1} is a tableau constructed for c following the rules in Def. 12, then $\{\langle T_{i_1},\_,\_,\_\rangle\}$ is a nested tableau for c.
- **Nesting (N)**: If NT is a nested tableau for c, then $NT'=NT\cup\{\langle T_k,n,B_n,\exists(a_n,c_n)\rangle\}$ is a nested tableau for c, if T_k is a tableau for c_n, B_n is an open branch in a tableau T_n in NT whose leaf is $\exists(a_n{:}P_n\to P_{n+1},c_n)$, and there is no other $\langle T_{k'},n,B_n,\exists(a_n,c_n)\rangle$ in NT.

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

Definition 19 (Open/Closed Nested Branch, Nested Tableau Refutation). A nested branch *NB* is *closed* if it contains $\exists(a,\texttt{false})$ or `false`; otherwise, it is *open*. A nested tableau is closed if *all* its nested branches are closed.

A *nested tableau refutation* for (the unsatisfiability of) a graph property c in *NCNF*, is a closed nested tableau NT for c built following the rules given in Def. 18.

Example 8. In Appendix B we show a nested tableau refutation for condition $c=c_1\wedge c_2\wedge c_3$, with $c_1=\exists(\emptyset\longrightarrow[\bullet\overset{x^+}{\Rightarrow}\bullet],\texttt{true})$, $c_2=\exists(\emptyset\longrightarrow[\bullet\overset{y^+}{\Rightarrow}\bullet],\texttt{true})$, and $c_3=\neg\exists(\emptyset\longrightarrow\left[\begin{smallmatrix}\bullet\xrightarrow{x}\bullet\\ \bullet\xrightarrow{y}\bullet\end{smallmatrix}\right],\texttt{true})$.

5. Soundness and Completeness of Nested Tableaux

Soundness is the property that ensures that what we can show with a given proof method is correct. More precisely, in our case, that we only refute properties that are

unsatisfiable. Conversely, completeness ensures that we can prove all consequences of any given set of formulas. In our case, that we can refute all properties that are unsatisfiable. Obviously, soundness and completeness depend on the class of models considered.

Sometimes no complete proof method exists for the intended class of models. This is what happens in our case for the class of models considered up to now. However, showing the method to be complete for a slightly different class of models illustrates nicely its power.

In this section, we first prove that our tableau method is sound for the class of models considered up to now. Then, after discussing why no complete method can exist in this case, we consider different classes of models as possible candidates and we show that our tableau method is sound and complete for the chosen class.

5.1. Soundness

In this subsection we prove that our tableau method is sound. In particular, soundness means that if we are able to construct a nested tableau where all its branches are closed then we are sure that our original condition c is unsatisfiable or, conversely, that if c is satisfiable any nested tableau for c includes an open nested branch. However, our Soundness Theorem below is more general than what one can expect. Instead of just proving that our method is sound for our class of models, we show that our method is sound for any class of models $\mathcal{M}$, provided that our basic tableau rules are sound for $\mathcal{M}$. This will simplify the proof of Theorem 2.

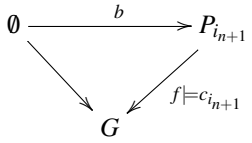
The proof of soundness uses Lemma 6 stating that, if the given rules used to construct a basic tableau are sound, then the tableau is sound, and Lemma 5 that shows the soundness of our rules. It also uses the fact that if all branches of the nested tableau are closed then it is finite. In particular, 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 show that NB can be extended by a branch of the new tableau using again Lemma 6.

Theorem 1 (Soundness) *Given a graph property c in NCNF, if the basic tableau rules given in Def. 12 are sound for a given class of models $\mathcal{M}$, then if there is a nested tableau refutation for c then c is unsatisfiable in $\mathcal{M}$.*

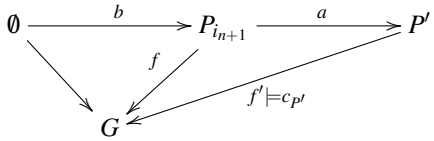
PROOF. First, notice that if the given basic tableau rules are sound, by Lemma 6, any basic tableaux is sound. Hence we may assume that all tableaux in NT are sound. Notice also that, if all nested branches in a nested tableau NT for c are closed, then NT is finite. The reason is that, on the one hand, closed nested branches are finite, since we only extend open nested branches; on the other hand, we know that the tableaux in NT are finite; therefore, by König's Lemma, NT is finite.

We will prove by induction on the number of tableaux in NT, that if c is satisfiable and the basic tableau rules are sound, then there is an open nested branch $NB = B_{i_1}, \ldots, B_{i_k}$ in NT, contradicting the hypothesis that NT is closed. Moreover, we will prove that there is a graph G such that, for every positive literal $\ell = \exists(a : P_{i_k} \to P', c_{P'})$ in B_{i_k}, $G \models \exists(b : \emptyset \to P', c_{P'})$, where $b = a \circ a_{i_1 i_k}$.

- The base case, when NT consists only of the initial tableau T_{i_1} is a direct consequence of Lemma 6.
- Let us suppose that $NT = NT' \cup \{\langle T, i_n, B_{i_n}, \exists(a_{i_n} : P_{i_n} \to P_{i_{n+1}}, c_{i_{n+1}})\rangle\}$. By inductive hypothesis, there is an open nested branch $NB = B_{i_1}, \ldots, B_{i_k}$ in NT'. More precisely, B_{i_k} must coincide with B_{i_n} since, otherwise, the open nested branch NB would also be in NT contradicting the hypothesis. This means that there is a graph G that, for every positive literal $\ell = \exists(b_{i_n} : P_{i_n} \to P', c_{P'})$ in B_{i_n}, $G \models \exists(b : \emptyset \to P', c_{P'})$, where $b = b_{i_n} \circ a_{i_1 i_n}$. Then, in particular, in the case of the opener, $G \models \exists(b : \emptyset \to P_{i_{n+1}}, c_{i_{n+1}})$, where $b = a_{i_n} \circ a_{i_1 i_n}$, i.e., there exists f, such that:



According to the nesting rule, T is a tableau for $c_{i_{n+1}}$, and we know that $f \models c_{i_{n+1}}$, thus there must be an open branch B in T such that f satisfies all the literals in the branch including every positive literal $\exists(a : P_{i_{n+1}} \to P', c_{P'})$. Hence, there exists f', such that:



i.e., $G \models \exists(a \circ b : \emptyset \to P', c_{P'})$ ■

5.2. Choosing the Models for Completeness

In general, in most application areas of Computer Science where graphs play an important role, these graphs are finite. As a consequence, we could think that the adequate class of models for GNL should be the class of finite graphs. Unfortunately, as shown by Trakhtenbrot [19], any deduction method is incomplete, because finite graph satisfiability is not semi-decidable, which means that no complete deduction method can exist if models are just finite graphs. A similar problem exists when dealing with paths. Even if the class of models includes infinite graphs, we cannot expect completeness if we consider that path expressions denote (finite) paths in a given graph. The reason is that the satisfiability of a condition stating that there is no path between two given nodes is not semi-decidable. The problem with paths is similar to what happens in first-order logic (FOL), in the sense that FOL is not complete with respect to the class of finitely-generated models. However FOL is complete if models may include additional non finitely generated elements. In our case, finite paths are finitely generated by the edges of the graph so, to avoid incompleteness, we consider that the models of GNL are finite or infinite graphs, including some infinite paths, in addition

to the "standard" paths. More precisely, models are patterns, which we call *weak* ω-*closed patterns*, where every path expression corresponds to some (finite or infinite) paths in the pattern.

However, there is an additional technical problem. If, for instance, we are using regular expressions to specify path expressions and, in a given condition, a path expression between two nodes is labelled with x^+, then that path expression could not denote an infinite path, because the language specifying the path labels, $\{x, xx, xxx, \ldots, x^n, \ldots\}$, consists only of finite words, which means that the paths specified by that language must be finite. In other words, if we want to allow that path expressions may denote infinite paths in the models of GNL, the languages used in these expressions would need to include infinite words. More precisely, if a path expression is labelled by the language $\{x, xx, xxx, \ldots, x^n, \ldots\}$, we would need to assume that the language also includes an infinite sequence of $x's$, which is the ω-limit [20] of the sequence $x, xx, xxx, \ldots, x^n, \ldots$.

Then, if we want to work with this new class of models, we have to reconsider the properties that a class of labelling languages over Σ, $\mathcal{L}(\Sigma)$, must satisfy:

Definition 20 (Path Labelling Languages 2). Given a set of labels Σ, $\mathcal{L}^\infty(\Sigma)$ is a *class of path labelling ∞-languages over* Σ, if it satisfies the following conditions:

1. If $L \in \mathcal{L}^\infty(\Sigma)$ then $\emptyset \subset L \subseteq \Sigma^+ \cup \Sigma^\omega$.
2. If $L_1, L_2 \in \mathcal{L}^\infty(\Sigma)$ then $L_1 \cdot L_2 \in \mathcal{L}^\infty(\Sigma)$.
3. If $L \in \mathcal{L}^\infty(\Sigma)$ then $L = S_0 \cup (\bigcup_{x \in S_1} \{x \cdot s |\ s \in L^x\})$ for some finite sets of labels $S_0, S_1 \subseteq \Sigma$, such that,
   - $\{x\} \in \mathcal{L}^\infty(\Sigma)$ for every $x \in S_0$, and
   - $L^x \in \mathcal{L}^\infty(\Sigma)$ and $\{x\} \cdot L^x \in \mathcal{L}^\infty(\Sigma)$ for every $x \in S_1$.
4. For each $s \in \Sigma^\omega$, if $L \in \mathcal{L}^\infty(\Sigma)$ and $\texttt{Pref}(s) \cap L$ is an infinite set, then $s \in L$, where $\texttt{Pref}(s)$ is the set of finite prefixes of s.

Notice that condition 3 is the same as in Def. 3, even if languages may include infinite words. Condition 4 is new and requires the languages of the class to be closed under ω-limits [20].

Again, as in Def. 3, the set $d_L = \{\{x\} |\ x \in S_0\} \cup \{\{x\} \cdot L^x |\ x \in S_1\} \subseteq \mathcal{L}^\infty(\Sigma)$ is the unique finite disjunctive decomposition of L.

Definition 21 (Weak ω-Closed Patterns). Let $\prec$ be the *prefix order* on path expressions, defined as: for all $\langle n, L_1, n_1\rangle, \langle n, L_2, n_2\rangle$, $\langle n, L_1, n_1\rangle \prec \langle n, L_2, n_2\rangle$ if there is $\langle n_1, L', n_2\rangle$ such that $L_2 = L_1 \cdot L'$.

Then, a pattern P is *weak* ω-*closed* if for every $\langle n, L, n'\rangle \in \Rightarrow_P$ there is a string $s \in L$ such that one of the following two conditions holds:

- $s \in \Sigma^+$ and $\langle n, \{s\}, n'\rangle \in \rightarrow^+_{G_P}$
- $s \in \Sigma^\omega$ and there is an infinite set $S = \{\langle n, \{s_i\}, n_i\rangle\}_{i \geq 0}$, such that, $S \subseteq \rightarrow^+_{G_P}$ and, for each i, $\langle n, \{s_i\}, n_i\rangle \prec \langle n, \{s_{i+1}\}, n_{i+1}\rangle$ and $s_i \in \texttt{Pref}(s)$.

The intuition of this definition is that, as discussed above, all path expressions in this pattern correspond to some finite or infinite path. More precisely, we require that

for every path expression $\langle n, L, n' \rangle$ either G_P includes a finite path $\langle n, \{s\}, n' \rangle$, with $s \in L$, or G_P includes a sequence of increasing paths such that its limit s is in L.

So, from now on, we will consider that the models of a set of conditions are weak ω-closed patterns, i.e., graphs that may include some infinite paths, as long as they do not contradict the negative conditions included in the set. In particular, as we can see in our completeness proof (cf. Sect. 5.3), when we can not refute a set (or, rather, a conjunction) of conditions it is because we can find some models, perhaps including some infinite paths required by the given positive conditions, that do also satisfy the given negative conditions. Specifically, in Lemma 9 we prove that, if we can not refute a graph property, then we can prove the existence of a model (cf. Def. 21) of the condition which is a weak ω-closed pattern.

It is straightforward to show that our tableau method is still sound for this new class of models:

Theorem 2 (Soundness) *Given a graph property c in NCNF, if there is a nested tableau refutation for c, then there is no weak* ω*-closed pattern satisfying c.*

PROOF. According to Theorem 1 we only need to prove that our basic tableau rules are sound for the new class of models. But this is straightforward:

- (init): For every f, $f \models \texttt{true}$.
- (ext) As in the proof of Lemma 5 if $\ell_1 \vee \cdots \vee \ell_n$ is a clause in c_P, then if $f \models c_P$ implies $f \models \ell_1 \vee \cdots \vee \ell_n$.
- (L) and (PL). Lemma 1 was proved for all pattern morphisms, which means that Lemmas 2 and 3 are also satisfied for all pattern morphisms. Hence, the two rules are sound.
- (SR) If a is split mono, according to 2. in Remark 3, $\ell = \exists(a : P \to Q, \texttt{true})$, is a tautology for all pattern morphism. Hence, the rule is also sound.
- (U) By Prop. 5, $\bigvee_{u \in \texttt{Unfold}(P, \langle n, L, n' \rangle)} \exists(u, \texttt{true})$ is a tautology also for path expressions including infinite strings and the new class of models, because of the requirements for path labelling languages including infinite strings (cf. Def. 20), which implies that this rule is also sound. ■

5.3. A Completeness Proof

To prove completeness of our tableau method, we need a notion of *saturation* of nested tableaux, describing some kind of fairness that ensures that we do not postpone indefinitely an inference step. In addition to assuming that all the basic tableaux included in the given nested tableau are semi-saturated, this implies two issues: the choice of the hook for each tableau and the exhaustive application of all possible unfoldings. Roughly, if a (positive) literal is never chosen as a hook we will be unable to make inferences between the inner condition of that literal and other literals, because the inner condition will never be disclosed. Similarly, if some possible unfolding is never performed, we may fail to see a contradiction between some conditions.

Let us see how conditions evolve along a nested branch when all basic tableaux are semi-saturated. If ℓ_i is a literal in a branch B_i, included in the nested branch $NB =$

$B_0,\ldots,B_i,\ldots,B_j,\ldots$ in NT, with contexts $\emptyset \xrightarrow{a_0} \ldots P_i \xrightarrow{a_i} \cdots \xrightarrow{a_{j-1}} P_j,\ldots$, if ℓ_i is not chosen as the hook for that branch, it will be transformed, via a lift rule, into the condition $|\texttt{Shift}(a_i,\ell_i)|$ that, by semi-saturation, will be part of the inner condition of the opener for the next tableau. If $\texttt{Shift}(a_i,\ell_i) = \exists(b_i,d_i)$ (resp. $\texttt{Shift}(a_i,\ell_i) = \neg\exists(b_i,d_i)$) and b_i is not an isomorphism, $|\texttt{Shift}(a_i,\ell_i)|$ will be the literal $\ell_{i+1} = \exists(b_i,|d_i|)$ (resp. $\ell_{i+1} = \neg\exists(b_i,|d_i|)$) that will be part of the next branch, B_{i+1}. Otherwise, if b_i is an isomorphism, $|\texttt{Shift}(a_i,\ell_i)|$ will be a condition with nesting level strictly smaller than the nesting level of ℓ_i and, in particular, the inner condition of ℓ_i would be disclosed when computing $|\texttt{Shift}(a_i,\ell_i)|$. Applying this argument several times, we would have $\ell_j = |\texttt{Shift}(a_{j-1}\circ\cdots\circ a_i,\ell_i)| = |\texttt{Shift}(a_{ij},\ell_i)|$ in B_j, unless $\texttt{Shift}(a_{ij},\ell_i) = \exists(b_j,d_j)$ (resp. $\texttt{Shift}(a_i,\ell_i) = \neg\exists(b_j,d_j)$) and b_j is an isomorphism. Moreover, by Lemma 1, we know that $f \models \ell_j$ if, and only if, $f\circ a_{ij} \models \ell_i$.

Definition 22 (Saturation). A nested tableau NT is *saturated* if all the tableaux included in NT are semi-saturated and, for each nested branch $NB = B_0,B_1,\ldots,B_j,\ldots$ in NT either NB is closed or the following conditions hold:

1. For each positive literal $\exists(a_i,c_i)$ in NB with nesting level k on branch B_i in tableau T_i one of the following conditions holds:
   - $\exists(a_i,c_i)$ is a hook for B_i and the leaf of B_i is a tableau opener.
   - There is some $j > i$ such that if $\texttt{Shift}(a_{ij},\ell_i) = \exists(b_j,d_j)$, either b_j is an isomorphism or $\exists(b_j,d_j)$ is the hook in B_j.
2. For each path expression $\langle n,L,n'\rangle \in \Rightarrow_{P_i}$ in the context P_i of B_i, there is a $j \geq i$ such that the corresponding unfolding rule (U) has been applied in B_j over a path expression $\langle a_{ij}(n),L,a_{ij}(n')\rangle \in \Rightarrow_{P_j}$.

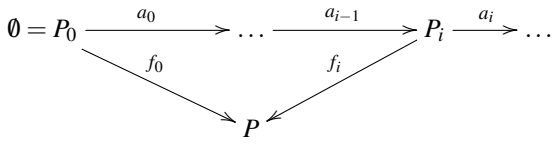
Lemma 8 (Existence of Saturated Nested Tableau) *Given a graph property c in NCNF, there exists a saturated nested tableau NT for c.*

PROOF. We can build this tableau as follows. First, according to Lemma 4, we build a semi-saturated finite tableau T_{i_1} for c. Then, for every open branch in T_{i_1} we use its leaf as an opener to build a new semi-saturated tableau. We continue this process for all open branches of these tableaux, and so on. This process may never end, so in the limit, we may have an infinite number of tableaux. Then, by construction, all the tableaux in NT are semi-saturated. To ensure that the other properties are satisfied, on the one hand, we have to use a fair strategy in the selection of hooks. This can be done by having, for each nested branch, a queue that includes the literals that are pending to be chosen as hooks. Similarly, to ensure that all unfoldings are performed, we may also keep queues of pending unfoldings. So, when opening a new tableau for a given nested branch, we would choose the hook for that tableau according to the former queue, and we would perform an unfolding according to the latter queue, which is equivalent to adding to the opener the corresponding unfolding tautology. ■

When working with tableaux, the standard way to show completeness is to prove that open branches in saturated tableaux define models of the given formulas. In our case, this is shown in the lemma below. In particular, in our case, we prove that, for any

open nested branch NB in a saturated nested tableau NT, the colimit of the morphisms associated with NB is a weak ω-closed pattern P that satisfies NT.

Lemma 9 (Models of Open Nested Branches) *Let NB be an open nested branch in a saturated nested tableau NT for a graph property c in NCNF, let $\emptyset \xrightarrow{a_0} \dots P_i \xrightarrow{a_i} \cdots \xrightarrow{a_{j-1}} P_j, \dots$ be the corresponding sequence of contexts for NB, and let P be its colimit, then P is a weak ω-closed pattern and $P \models c$.*



PROOF SKETCH. *(See Appendix A.9 for the detailed proof)* The proof of this lemma has two parts. In the first part we prove that P is a valid model, i.e., it is a weak ω-closed pattern. In particular, we show that if P includes a path expression $\langle n, L, n' \rangle$, the successive application of unfolding rules, which is guaranteed by the saturation of the given nested tableau, will ensure that there is a path in P matching that path expression.

In the second part we prove that P satisfies the given condition c. Specifically, we show by induction on the nesting level of literals that, for every literal ℓ in a branch B_i in NB, $f_i \models \ell$. In particular, f_0 satisfies all the literals in the first branch of NB implying, by Lemma 7, that P satisfies c. This proof considers three different cases.

In the first case we assume that there is some $j > i$, where $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level smaller than $n+1$. For example, this happens when $\texttt{Shift}(a_{ij}, \ell)$ is a condition of the form $\exists(a'_j, d_j)$ or $\neg\exists(a'_j, d_j)$ and a'_j is an isomorphismm. The proof of this case is based essentially on the inductive hypothesis.

Secondly, we consider the case where ℓ is positive and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$. In this case, the proof makes use of the Shifting Lemma (Lemma 1), the saturation of the nested tableau, the inductive hypothesis and the fact that if $\ell = \exists(a_i : P_i \to P_{i+1}, c_{P_{i+1}})$ is taken as a hook, then the next context will be P_{i+1} and $f_{i+1} \circ a_i = f_i$.

Finally, we consider the case where ℓ is negative, $\ell = \neg\exists(a'_i : P_i \to Q_i, d_i)$, and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$. In this case, we have to prove that if there is a morphism $f : Q_i \to P$ satisfying $f \circ a'_i = f_i$, then f does not satisfy d_i. The proof makes use of the FinCol property (see Prop. 2) of `Pattern` that ensures that if there is such a morphism f there will be a P_j and a morphism $Q_i \to P_j$ that allow us to apply the partial lift rule. Moreover, that rule will be applied because if NT is saturated, T_j will be semi-saturated, which means that all possible partial lift rules will be applied to the hook in B_j. At this point, the Shifting Lemma, together with the inductive hypothesis, will ensure that f does not satisfy d_i. ■

Finally, we have all the ingredients to prove completeness.

Theorem 3 (Completeness) *Given a graph property c in NCNF, if c is unsatisfiable, then there is a nested tableau refutation for c.*

PROOF. Because of Lemma 8 we know that there exists a saturated nested tableau NT for c. But if NT is not a tableau refutation for c, there exists at least one open nested branch NB in NT. Then, by Lemma 9 we know that there is a weak ω-closed pattern P, such that $P \models c$. ■

A direct consequence of this theorem is that, if we restrict GNL to conditions not including paths in patterns (i.e., patterns are just graphs), the same inference rules, excluding unfolding, are complete for the class of finite and infinite graphs. The reason is that, first, if patterns include no paths then unfolding is useless and, second, given $P = (G_P, \Rightarrow_P)$ and $Q = (G_Q, \Rightarrow_Q)$, if $\Rightarrow_P = \emptyset$, we have that $f : P \to Q$ is a pattern morphism if and only if $f : G_P \to G_Q$ is a graph morphism. As a consequence, considering the role of morphisms in the definition of satisfaction, we have that for any weak ω-closed pattern P and any graph condition c including no path expressions in its patterns, we have that $P \models c$ if and only if $(G_P, \emptyset) \models c$. This result, generalizes the completeness proof in [7] to the case where morphisms in conditions and in the definition of satisfaction are not necessarily monomorphisms.

Corollary 1 (Completeness) *The nested tableau method is complete with respect to the class of graphs, for graph conditions whose patterns include no path expressions.*

6. Related Work

The idea of expressing graph properties by means of graphs and graphs morphisms has its origins in the notions of graph constraints and application conditions [21, 22, 23]. In [24], Rensink presented a logic for expressing graph properties, closely related with the Logic of Nested Graph Conditions (LNGC) of Habel and Penneman [3]. Moreover, in [25], Habel and Radke, presented a notion of HR^+ conditions with variables that allowed them to express properties about paths, but no deduction method was presented. First approaches to provide deductive methods to this kind of logics were presented in [15] for a fragment of LNGC, and by Pennemann [18, 4] for the whole logic. Unfortunately, Penneman was unable to show the completeness of his approach. In [26], Poskitt and Plump propose an extension of nested conditions with monadic second-order (MSO) properties over nodes and edges. In particular, they can define path predicates that allow for the direct expression of properties about arbitrary-length paths between nodes. However, again this formalism lacks a deduction method. Lambers and Orejas [7] defined the nested tableaux method restricted to LNGC and were able to show the completeness of a subset of Pennemann's inference rules. On the other hand, Navarro, Orejas and Pino [10] presented a complete proof system for reasoning about XML patterns, including paths. Our work here extends [7, 10], but this extension, as discussed in the introduction, is far from straightforward.

The work in this paper is also related to foundational work in the area of query languages for graph databases. An excellent survey has been published recently summarizing most of this work [27]. However, even if conceptually there is a close connection, the work conducted in this area has little relation to what we present here. On the one hand, that work concentrates on the logic fragment associated to some classes of

graph queries. Being precise, in our terms, they essentially work with formulas with at most one level of nesting. On the other hand, they mainly concentrate on algorithmic, complexity and decidability issues (see, e.g., [28, 29, 30, 31, 32]).

7. Conclusion and Outlook

In this paper we have presented GNL, an extension of LNGC, including the possibility of specifying navigational properties, and we have presented a sound and complete tableau refutation method for this logic. Moreover, using a-satisfaction [3], instead of m-satisfaction which is the common choice, allows us to avoid exponential branching in tableau refutations. More precisely, the result of shifting a literal in our context is just another literal but when using m-satisfaction, the result of shifting a literal is, in the worst case, an exponential disjunction of literals. In addition, the use of a-satisfaction simplified considerably our technical constructions, including the completeness proofs.

GNL is not restricted to a specific class of graphs, but it can be used for any class that satisfies the conditions stated in the paper. Moreover, in [11] we presented basic ideas for a general formulation of GNL, applicable to any category of structures satisfying some general requirements, and in [9] we have shown that LNGC can be used to reason about attributed graphs, and we believe that it would not be difficult to show that this is also the case for GNL.

As said in the introduction, we think that GNL can be useful in many areas of Computer Science. However we are specifically interested in the areas of model-driven development (MDD) and graph databases.

In MDD systems are described by models, typically graphical models, like the different UML diagrams, Petri Nets or the various kinds of transition diagrams, and development is expressed by model transformations. More precisely, when modeling a system, besides the diagrams, we may have to declare some constraints. Most often, this is done using some textual language, like OCL in the case of UML, or directly first-order logic. We believe that expressing these constraints in LNGC or in GNL if navigational properties are required, allows one to express these constraints in a simpler and more intuitive way, since the logical notation uses directly the elements of the modeling notation. This is especially the case when the constraints refer to the structure of the models where, if using a textual language, we would need to encode that structure.

Moreover, when describing the methods of a model using graph transformation, which is something quite natural if computation states are modelled by graphs, the use of a graph logic to specify constraints provides some powerful techniques for verification (see, for instance, [3]). A special case, in this sense is the work of Parisi-Presicce, Koch and Mancini on the specification and analysis of access control policies (see, e.g. [33]). In particular, they specify access control policies using graph constraints (i.e., properties expressed similarly as the conditions in LNGC or GNL) to describe the valid states of a system, and graph transformation rules to specify operations. Interestingly, they use some form of ad-hoc deduction on these constraints to check the consistency of a policy.

In the area of model transformation, conditions defined using GNL can be used to state that some related models are synchronized. More precisely, model synchronization is a difficult problem that is relevant to different areas of Computer Science, i.e., MDD (see, for instance, [34]), programming languages (see, for instance, [35]), and databases (see, for instance, [36]). This problem can be explained as follows. We have two (or more) models M_1 and M_2, where M_2 has been obtained from M_1 by some transformation. For some reason, either M_1 or M_2 have been modified afterwards, how can we modify the other model so that one model continues being the transformation of the other? If the given classes of models are represented by some kind of graphs, as is often the case in MDD, this problem can be formulated as the problem of repairing a graph that does not satisfy a set of constraints, where these constraints describe what it means that a model is the transformation of another model. In particular, our work [37], was partially motivated by this problem.

We also believe that GNL could be used as a logical foundation for graph databases and their query languages. One could argue that GNL graphs do not include any data (except labels), but as said above this is not exactly true. GNL is a general logic that may be used to reason about many classes of graphs, as long as their associated categories satisfy the requirements stated in Def. 1. Specifically, in [9] LNGC is defined over the category of symbolic graphs [38], where a symbolic graph is an attributed graph where, in addition, we can express constraints on its attributes. More precisely, a symbolic graph, (G,Φ), consists of an attributed graph G whose values are represented by variables, and of a set of constraints Φ over a given data algebra, specifying the possible values of these variables[4].

In particular, we think that GNL can play a role in connection with queries and with integrity constraints. With respect to queries, we have checked that all queries in the LDBC benchmark [39], except counting queries, can be easily expressed in GNL when defined over symbolic graphs. Actually, in Ex. 4, we present a very simple example of a boolean query and how it could be solved using our tableau method. In particular, the database G would be represented by the condition $\exists(\emptyset \to G, \texttt{true})$ and, to solve a query q, we would refute q with respect to the database. Then, the completeness of our tableau method would guarantee some sort of completeness of the answers obtained from each closed branch of the tableau.

It is clear that, in general, queries belong to a very simple kind of graph conditions and, therefore, that our results may be considered *too powerful*. However, besides the possibility of defining specialized proof methods for the *standard* class of queries, our approach would allow us to explore other kind of situations like, for instance, to study possible ways of defining deductive graph databases.

With respect to integrity constraints, not only we can use our approach to state them, but we can also use it to check if a given database satisfies them, and to repair it, in case it does not satisfy the given constraints. In particular, as said above, in [37], using our tool AutoGraph, we show how we can repair a graph that does not satisfy a set of graph conditions (without path expressions). It is part of future work to extend

[4]For instance, these constraints may just be equality constraints, $X = v$, stating that the value of X should be v.

AutoGraph and this automatic repair approach also to GNL.

References

[1] B. Courcelle, The expression of graph properties and graph transformations in monadic second-order logic, in: G. Rozenberg (Ed.), Handbook of Graph Grammars, World Scientific, 1997, pp. 313–400.

[2] L. Cardelli, P. Gardner, G. Ghelli, A spatial logic for querying graphs, in: P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. J. Eidenbenz, R. Conejo (Eds.), Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings, Vol. 2380 of Lecture Notes in Computer Science, Springer, 2002, pp. 597–610.

[3] A. Habel, K.-H. Pennemann, Correctness of high-level transformation systems relative to nested conditions, Mathematical Structures in Computer Science 19 (2) (2009) 245–296.

[4] K.-H. Pennemann, Development of Correct Graph Transformation Systems, PhD Thesis, Department of Computing Science, Univ. of Oldenburg, 2009.

[5] P. Baumgartner, A. Fuchs, C. Tinelli, Implementing the model evolution calculus, International Journal on Artificial Intelligence Tools 15 (1) (2006) 21–52.

[6] A. Riazanov, A. Voronkov, The design and implementation of VAMPIRE, AI Commun. 15 (2-3) (2002) 91–110.

[7] L. Lambers, F. Orejas, Tableau-based reasoning for graph properties, in: Graph Transformation - 7th International Conference, ICGT 2014, Vol. 8571 of Lecture Notes in Computer Science, Springer, 2014, pp. 17–32.

[8] S. Schneider, L. Lambers, F. Orejas, Symbolic model generation for graph properties, in: Fundamental Approaches to Software Engineering FASE 2017, Held as Part of ETAPS 2017, Vol. 10202 of Lecture Notes in Computer Science, Springer, 2017, pp. 226–243.

[9] S. Schneider, L. Lambers, F. Orejas, Automated reasoning for attributed graph properties, Int. J. Software Tools for Technology Transfer 20 (4) (2018) 705–737. URL https://doi.org/10.1007/s10009-018-0496-3

[10] M. Navarro, F. Orejas, E. Pino, Satisfiability of constraint specifications on XML documents, in: N. Martí-Oliet, P. C. Ölveczky, C. L. Talcott (Eds.), Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday, Vol. 9200 of Lecture Notes in Computer Science, Springer, 2015, pp. 539–561.

[11] L. Lambers, M. Navarro, F. Orejas, E. Pino, Towards a navigational logic for graphical structures, in: Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig, Vol. 10800 of Lecture Notes in Computer Science, Springer, 2018, pp. 124–141.

[12] F. Orejas, E. Pino, M. Navarro, L. Lambers, Institutions for navigational logics for graphical structures, Theoret. Comput. Sci. 741 (2018) 19–24.

[13] J. Goguen, R. Burstall, Institutions: Abstract model theory for specification and programming., J. ACM 1 (39) (1992) 95–149.

[14] D. Sannella, A. Tarlecki, Specifications in an arbitrary institution, Inf. Comput. 76 (2/3) (1988) 165–210.

[15] F. Orejas, H. Ehrig, U. Prange, Reasoning with graph constraints, Formal Aspects of Computing 22 (3-4) (2010) 385–422.

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

[17] R. Hähnle, Tableaux and related methods, in: J. A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Elsevier and MIT Press, 2001, pp. 100–178.

[18] K.-H. Pennemann, Resolution-like theorem proving for high-level conditions, in: Graph Transformations, 4th International Conference, ICGT 2008, Vol. 5214 of Lecture Notes in Computer Science, Springer, 2008, pp. 289–304.

[19] B. A. Trakhtenbrot, The impossibility of an algorithm for the decision problem on finite classes, Doklady Akademii Nauk SSSR 70 (23) (1950) 569–572.

[20] L. Staiger, ω languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3: Beyond Words, Springer-Verlag New York, Inc., New York, NY, USA, 1997, pp. 339–387.

[21] H. Ehrig, A. Habel, Graph grammars with application conditions, in: G. Rozenberg, A. Salomaa (Eds.), The Book of L, Springer, Berlin, 1986, pp. 87–100.

[22] R. Heckel, A. Wagner, Ensuring consistency of conditional graph rewriting - a constructive approach, Electr. Notes Theor. Comput. Sci. 2 (1995) 118–126.

[23] A. Habel, R. Heckel, G. Taentzer, Graph grammars with negative application conditions, Fundamenta Informaticae 26 (1996) 287–313.

[24] A. Rensink, Representing first-order logic using graphs, in: Graph Transformations, Second International Conference, ICGT 2004, Vol. 3256 of Lecture Notes in Computer Science, Springer, 2004, pp. 319–335.

[25] A. Habel, H. Radke, Expressiveness of graph conditions with variables, ECEASST 30.

[26] C. M. Poskitt, D. Plump, Verifying monadic second-order properties of graph programs, in: Graph Transformation - 7th International Conference, ICGT 2014, Vol. 8571 of Lecture Notes in Computer Science, Springer, 2014, pp. 33–48.

[27] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, D. Vrgoč, Foundations of modern query languages for graph databases, ACM Comput. Surv. 50 (5) (2017) 68:1–68:40.

[28] P. Barceló, L. Libkin, J. L. Reutter, Querying graph patterns, in: Proceedings of the 30th ACM PODS 2011, 2011, pp. 199–210.

[29] P. T. Wood, Query languages for graph databases, SIGMOD Record 41 (1) (2012) 50–60.

[30] P. Barceló, Querying graph databases, in: Proceedings of the 32nd ACM PODS 2013, 2013, pp. 175–188.

[31] P. Barceló, L. Libkin, J. L. Reutter, Querying regular graph patterns, J. ACM 61 (1) (2014) 8:1–8:54.

[32] P. Barceló, M. Romero, M. Y. Vardi, Semantic acyclicity on graph databases, SIAM J. Comput. 45 (4) (2016) 1339–1376.

[33] M. Koch, L. V. Mancini, F. Parisi-Presicce, Graph-based specification of access control policies, J. Comput. Syst. Sci. 71 (1) (2005) 1–33.

[34] H. Giese, R. Wagner, From model transformation to incremental bidirectional model synchronization, Software and System Modeling 8 (1) (2009) 21–43.

[35] M. Hofmann, B. C. Pierce, D. Wagner, Symmetric lenses, in: POPL 2011, ACM, 2011, pp. 371–384.

[36] C. J. Date, View Updating and Relational Theory, O'Reilly, 2012.

[37] S. Schneider, L. Lambers, F. Orejas, A logic-based incremental approach to graph repair, in: Fundamental Approaches to Software Engineering, Held as Part of ETAPS 2019, Vol. 11424 of Lecture Notes in Computer Science, Springer, 2019, pp. 151–167.

[38] F. Orejas, Symbolic graphs for attributed graph constraints, J. Symb. Comput. 46 (3) (2011) 294–315.

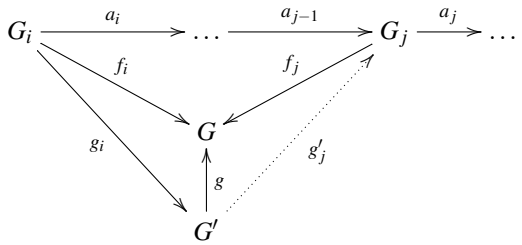
[39] The Linked Data Benchmark Council (LDBC), Social network benchmark, Tech. rep. (2017).
URL `https://github.com/ldbc/ldbc_snb_docs`

Appendix A. Detailed proofs

Appendix A.1. Proposition 1.

ELGraph *is cocomplete, has an initial object* $\emptyset$ *and satisfies the property FinCol.*

PROOF. We have to prove that ELGraph is cocomplete, has an initial object $\emptyset$ and satisfies the property FinCol, i.e., if $\{G_i \xrightarrow{a_i} G_{i+1}\}_{i\in\mathbb{N}}$ is an infinite sequence of morphisms between finite graphs where $\{G_i \xrightarrow{f_i} G\}_{i\in\mathbb{N}}$ in Graph is their colimit, for any i and any finite graph G', if there are two morphisms $g_i : G_i \to G'$ and $g : G' \to G$, satisfying $g \circ g_i = f_i$, then there is some $j \geq i$ and a morphism $g'_j : G' \to G_j$ such that $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$, where a_{ij} denotes the morphism $a_{j-1} \circ \cdots \circ a_i$.



The initial object is the empty graph $\emptyset$, since for every graph G in the category the empty morphism $e_G : \emptyset \to G$ is the unique morphism from $\emptyset$ to G.

The construction of colimits for set-based categories is quite standard. Let D be a diagram in the category of labelled graphs. Let $\equiv_D$ be the smallest equivalence relation over the elements (nodes and edges) of the graphs in D satisfying, for all morphisms $h : H \to H'$ in D, and all elements x, x' in H and H', respectively, that $x \equiv_D x'$ if $h(x) = x'$. Then we define G together with the family of morphisms $f_H : H \to G$, the colimit of D, as follows:

- $Node_G = \cup_{H\in D} Node_H / \equiv_D$.
- $Edge_G = \cup_{H\in D} Edge_H / \equiv_D$.
- $s_G(|e|) = s_H(e)$, $t_G(|e|) = t_H(e)$, $Label_G(|e|) = Label_H(e)$ if $e \in Edge_H$.
- For every H in D and every element (node or edge) x in H, $f_H(x) = |x|$.

It is not difficult to prove that G together with the family of morphisms f_H is indeed a colimit.

To prove FinCol, first notice that according to how colimits are defined, since $\{f_k : G_k \to G\}_{k\geq 0}$ is the colimit of the diagram $D = \{G_k \xrightarrow{a_k} G_{k+1}\}_{k\geq 0}$, it means that for every $k \geq 0$ and every $x_k \in G_k$, $f_k(x_k)$ is defined as $|x_k|$ for $\equiv_D$. Consider the given G_i in the sequence and the finite graph G' where $g_i : G_i \to G'$, $g : G' \to G$, and $g \circ g_i = f_i$. Given an element $x \in G'$, since $g(x)$ is an element in the colimit G, by construction of the colimit and by definition of $\equiv_D$, there must exist some k such that there is a unique $x_k \in G_k$ with $g(x) = |x_k|$ (we only need to go forward in the sequence to obtain this unicity). The same happens for every other element (node, edge or label) in G'. Then, as $Node_{G'}$, $Edge_{G'}$ and the set of labels that label an edge in G' are finite sets,

there must exist a graph G_j in the sequence (for instance, j being the maximum of all considered k) with the property that for every $x \in G'$ there is a unique $x_j \in G_j$ such that $|x_j| = g(x)$. This means that the graph morphism $g'_j : G' \to G_j$ defined: for all $x \in G'$, $x_j \in G_j$, $g'_j(x) = x_j$ if $g(x) = |x_j|$, is well-defined and such that $f_j \circ g'_j = g$.

To complete the proof, let us see that $g'_j(g_i(x_i)) = a_{ij}(x_i)$, holds for every $x_i \in G_i$. By definition of g'_j, $g'_j(g_i(x_i))$ is the unique element in G_j such that its class is $g(g_i(x_i))$, therefore $g'_j(g_i(x_i)) = a_{ij}(x_i)$ since $|a_{ij}(x_i)| = f_j(a_{ij}(x_i)) = f_i(x_i) = g(g_i(x_i))$. ■

Appendix A.2. Proposition 2.

Patterns and pattern morphisms form the category Pattern*. Moreover, if* Graph *has initial objects, is cocomplete and satisfies the property FinCol, then so does* Pattern*.*

PROOF. The existence of identity morphisms and morphism composition in Pattern is an almost direct consequence of the fact that pattern morphisms are morphisms in Graph, and the same applies to the associativity of morphism composition.

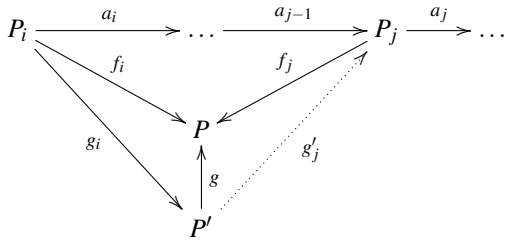
As in the case of graphs, the empty pattern is the initial object in Pattern, since for every pattern P the inclusion $\emptyset \hookrightarrow P$ is the unique morphism from $\emptyset$ to P.

Let D be a diagram in Pattern and let $D|_{\texttt{Graph}}$ be the corresponding diagram in Graph (i.e., the diagram consisting of morphisms $h : G_P \to G_{P'}$ for every morphism $h : P \to P'$ in D). Let $C = \{f_P : G_P \to H\}_{P \in D}$ be the colimit of $D|_{\texttt{Graph}}$ in Graph. Finally, let Q be the pattern $Q = (H, \Rightarrow_Q)$, where:

$$\Rightarrow_Q = \{\langle f_P(n), L, f_P(n')\rangle \mid f_P \in C \text{ and } \langle n, L, n'\rangle \in \Rightarrow_P\}$$

It is easy to see that $C_{\texttt{Pattern}} = \{f_P : P \to Q\}_{f_P \in C}$ is the colimit of D in Pattern. On the one hand, by definition, every $f_P \in C_{\texttt{Pattern}}$ is a morphism in Pattern, and by construction $C_{\texttt{Pattern}}$ is a cocone. On the other hand, let us suppose that $C' = \{f'_P : G_P \to Q'\}_{P \in D}$ is a cocone in Pattern. Then, $C'|_{\texttt{Graph}}$ is a cocone in Graph and, hence, there is a unique morphism $h : G_Q \to G_{Q'}$ such that $h \circ f_P = f'_P$, for every $P \in D$. But $h : Q \to Q'$ is also a morphism in Pattern, since if $\langle n, L, n'\rangle \in \Rightarrow_Q$ then, by definition, there is a $P \in D$ such that $\langle m, L, m'\rangle \in \Rightarrow_P$, $n = f_P(m)$ and $n' = f_P(m')$. But this means that $\langle f'_P(m), L_P, f'_P(m')\rangle \in \Rightarrow_{\widehat{Q'}}$, for some non-empty $L_P \subseteq L$. Hence, $\langle h \circ f_P(m), L_P, h \circ f_P(m')\rangle \in \Rightarrow_{\widehat{Q'}}$, i.e., $\langle h(n), L_P, h(n')\rangle \in \Rightarrow_{\widehat{Q'}}$

Finally, Pattern satisfies the property FinCol, i.e., given an infinite sequence of morphisms $\{P_i \xrightarrow{a_i} P_{i+1}\}_{i \geq 0}$ between finite patterns (i.e., their sets of nodes, edges and their set of path expressions are finite) and given their colimit $\{P_i \xrightarrow{f_i} P\}_{i \geq 0}$ in Pattern, for all morphisms $g_i : P_i \to P'$, $g : P' \to P$ in Graph, such that P' is finite and $g \circ g_i = f_i$, there is some $j \geq i$ and a morphism $g'_j : P' \to P_j$ such that $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$, where a_{ij} denotes the morphism $a_{j-1} \circ \cdots \circ a_i$.



On the one hand, according to how colimits in `Pattern` are defined, $\{f_j : G_{P_j} \to G_P\}_{i \geq 0}$ is a colimit in `Graph`, hence there must exist a k and a graph morphism $g'_k : G_{P'} \to G_{P_k}$ such that $f_k \circ g'_k = g$ and $g'_k \circ g_i = a_{ik}$. Notice that this also implies that for every $j > k$ there is a morphism $g'_j : G_{P'} \to G_{P_j}$ such that $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$.

On the other hand, we also know that

$$\Rightarrow_Q = \cup_{j \geq 0} \{\langle f_j(n), L, f_j(n') \rangle \mid \langle n, L, n' \rangle \in \Rightarrow_{P_j}\}$$

Let R_j be, for each j, the relation $R_j = \{\langle f_j(n), L, f_j(n') \rangle \mid \langle n, L, n' \rangle \in \Rightarrow_{P_j^+}\}$, then we have that $\Rightarrow_{\widehat{P}} = \cup_{j \geq 0} R_j$ and, for every j, $R_j \subseteq R_{j+1}$.

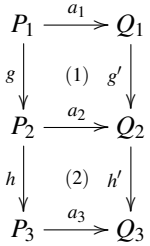
Finally, since P' is a finite pattern, there should be a k' such that, for every $\langle n, L, n' \rangle \in \Rightarrow_{P'}$, $\langle g(n), L, g(n') \rangle \in R_{k'}$. Let $j = max(k, k')$ then we have that the morphism $g'_j : P' \to P_j$ is a pattern morphism satisfying $f_j \circ g'_j = g$ and $g'_j \circ g_i = a_{ij}$ ■

Appendix A.3. Proposition 3.

Given morphisms $g : P_1 \to P_2$ and $h : P_2 \to P_3$ and given a condition c_{P_1}, we have:

$$\texttt{Shift}(h, \texttt{Shift}(g, c_{P_1})) = \texttt{Shift}(h \circ g, c_{P_1})$$

PROOF. By induction on the structure of conditions:



- $\texttt{Shift}(h, \texttt{Shift}(g, \texttt{true})) = \texttt{true} = \texttt{Shift}(h \circ g, \texttt{true})$
- $\texttt{Shift}(h, \texttt{Shift}(g, \exists(a_1 : P_1 \to Q_1, c_{Q_1}))) =$
  $\texttt{Shift}(h, \exists(a_2 : P_2 \to Q_2, \texttt{Shift}(g', c_{Q_1}))) =$
  $\exists(a_3 : P_3 \to Q_3, \texttt{Shift}(h', \texttt{Shift}(g', c_{Q_1})))$, by inductive hypothesis, $\texttt{Shift}(h', \texttt{Shift}(g', c_{Q_1})) = \texttt{Shift}(h' \circ g', c_{Q_1})$. Thus, $\exists(a_3 : P_3 \to Q_3, \texttt{Shift}(h', \texttt{Shift}(g', c_{Q_1}))) =$
  $\exists(a_3 : P_3 \to Q_3, \texttt{Shift}(h' \circ g', c_{Q_1})) =$
  $\texttt{Shift}(h \circ g, \exists(a_1 : P_1 \to Q_1, c_{Q_1}))$
- The other two cases hold directly by inductive hypothesis. ■

Appendix A.4. Proposition 4.

If $a: P \to Q$ is not split mono and $\Rightarrow_P = \emptyset$, then id_P satisfies $\neg\exists(a, c_Q)$, for any c_Q.

PROOF. First, notice that $P = (G_P, \emptyset)$. Then, we can see that $id_P \models \neg\exists(a, c_Q)$ if a is not split mono. Let us suppose that id_P satisfies $\exists(a, c_Q)$, then there exists a morphism $g: Q \to P$ such that $g \circ a = id_P$ and g satisfies c_Q. Thus, a would be split mono contradicting the assumptions. ■

Appendix A.5. Lemma 2.

Let $\ell_1 = \exists(a_1, c_1)$ and ℓ_2 be literals with morphisms $a_i: P \to Q_i$, for $i = 1, 2$. We define the lift of literals as follows:

$$\texttt{Lift}(\exists(a_1, c_1), \ell_2) = \exists(a_1, c_1 \wedge |\texttt{Shift}(a_1, \ell_2)|)$$

Then, $f \models \ell_1 \wedge \ell_2$ if, and only if, $f \models \texttt{Lift}(\ell_1, \ell_2)$.

PROOF. Assume $f: P \to G$ such that $f \models \exists(a_1, c_1 \wedge |\texttt{Shift}(a_1, \ell_2)|)$. That is, there exists a morphism $g: Q_1 \to G$ such that $f = g \circ a_1$ and $g \models c_1 \wedge |\texttt{Shift}(a_1, \ell_2)|$. Then, this is equivalent to $f \models \ell_1$ and $f \models \ell_2$, since by Lemma 1 we have that $g \circ a_1 \models \ell_2$. ■

Appendix A.6. Lemma 3.

Let $\ell_1 = \exists(a_1 : P \to Q_1, c_1)$ and $\ell_2 = \neg\exists(a_2 : P \to Q_2, c_2)$ such that there exists a morphism $g : Q_2 \to Q_1$ satisfying $a_1 = g \circ a_2$. We define the partial lift of literals as follows:

$$\texttt{PLift}(\exists(a_1, c_1), \ell_2) = \exists(a_1, c_1 \wedge |\texttt{Shift}(g, \neg c_2)|)$$

Then, $f \models \ell_1 \wedge \ell_2$ implies $f \models \texttt{PLift}(\ell_1, \ell_2)$.

PROOF. If $f \models \ell_1$, there is a morphism $h_1 : Q_1 \to G$ such that $f = h_1 \circ a_1 = h_1 \circ g \circ a_2$, and $h_1 \models c_1$. And if $f \models \ell_2$, there is no morphism $h_2 : Q_2 \to G$ with $f = h_2 \circ a_2$, and $h_2 \models c_2$. Then, since $f = h_1 \circ g \circ a_2$, we have $h_1 \circ g \models \neg c_2$, implying $h_1 \models |\texttt{Shift}(g, \neg c_2)|$ by Lemma 1. Since $h_1 \models c_1 \wedge |\texttt{Shift}(g, \neg c_2)|$ we conclude that $f \models \texttt{PLift}(\ell_1, \ell_2)$. ■

Appendix A.7. Lemma 4.

For any conditon c_P over P in NCNF and any natural number k, such that k is smaller or equal than the number of path expressions in $\Rightarrow_P$, there exists a finite semi-saturated tableau T for c_P including k unfolding tautologies.

PROOF. We can obtain finite semi-saturated tableaux as follows:

1. Apply the extension rule *(ext)* to all clauses in c_P (including the unfolding tautologies). Since this condition is a finite conjunction of finite clauses, we can apply this rule a finite number of times to include in the tableau all the literals in c_P. Then the resulting tableau will be finite.

2. For each branch B in the tableau:
   (a) If B only includes negative literals, there are no literals with split monomorphisms and $\Rightarrow_P = \emptyset$, then we have finished with this branch. Otherwise, apply the split reduction rule and the unfolding rule to k different path expressions in P, so that now B includes a positive literal.
   (b) If the branch is not closed, choose a hook, $\ell = \exists(a : P \to Q, c)$, and successively apply the lift rule, first to the hook and the first literal in the branch, next to the previous result and the second literal and so on, until we have applied it to all the literals in the branch. This is a finite process producing a finite branch whose final output is the literal

$$\exists(a,\ c \wedge \bigwedge_{\ell' \in B \setminus \{\ell\}} |\texttt{Shift}(a, \ell')|\ )$$

   (c) Successively apply, whenever is possible, the partial lift rule to the literal obtained previously and to all negative literals. This is also a finite process, producing the literal:

$$\exists(a,\ c \wedge \bigwedge_{\ell' \in B \setminus \{\ell\}} |\texttt{Shift}(a, \ell')| \wedge \bigwedge_{\ell' \in B^{PL}} |\texttt{Shift}(g, \neg d')|)$$

   with $B^{PL} = \{\neg\exists(a', d') \in B \setminus \{\ell\} \mid \exists g : a = g \circ a'\}$.

By construction, the tableau is semi-saturated. ■

Appendix A.8. Lemma 7.

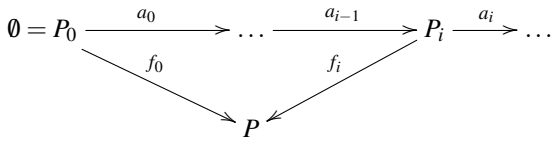
If T is a semi-saturated tableau for a condition c_P in NCNF, then we have:

1. *$f \models T$ implies $f \models c_P$.*
2. *For every $c'_P \subseteq c_P$ and every open branch B, if for every literal ℓ in B that is also in c'_P $f \models \ell$, then $f \models c'_P$.*
3. *If for every literal ℓ in B $f \models \ell$, then $f \models c_P$*

PROOF.
(2.) Since T is semi-saturated, B includes a literal for each clause in c'_P. Hence, if f satisfies all the literals in B that are also in c'_P, f satisfies each clause in c'_P and, so, it satisfies c'_P.
(3.) If condition (2.) holds for every $c'_P \subseteq c_P$, in particular, it also holds for c_P.
(1.) If $f \models T$ then there is an open branch B such that $f \models B$, since otherwise all branches will be unsatisfiable and so will be T. Hence f satisfies all the literals in B. Thus, by (3.) $f \models c_P$. ■

Appendix A.9. Lemma 9.

Let NB be an open nested branch in a saturated nested tableau NT for a graph property c in NCNF, let $\emptyset \xrightarrow{a_0} \ldots P_i \xrightarrow{a_i} \cdots \xrightarrow{a_{j-1}} P_j, \ldots$ *be the corresponding sequence of contexts for NB, and let P be its colimit, then P is a weak* ω*-closed pattern and* $P \models c$.



PROOF. As described in the sketch, the proof of this lemma has two parts. In the first part we prove that P is a valid model, i.e., it is a weak ω-closed pattern. In particular, we show that if P includes a path expression $\langle n, L, n' \rangle$, the successive application of unfolding rules, which is guaranteed by the saturation of the given nested tableau, will ensure that there is a path in P matching that path expression.

In the second part we prove that P satisfies the given condition c. Specifically, we show by induction on the nesting level of literals that, for every literal ℓ in a branch B_i in NB, $f_i \models \ell$. In particular, f_0 satisfies all the literals in the first branch of NB implying, by Lemma 7, that P satisfies c. This proof considers three different cases.

In the first case we assume that there is some $j > i$, where $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level smaller than $n+1$. For example, this happens when $\texttt{Shift}(a_{ij}, \ell)$ is a condition of the form $\exists(a'_j, d_j)$ or $\neg\exists(a'_j, d_j)$ and a'_j is an isomorphismm. The proof of this case is based essentially on the inductive hypothesis.

Secondly, we consider the case where ℓ is positive and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$. In this case, the proof makes use of the Shifting Lemma (Lemma 1), the saturation of the nested tableau, the inductive hypothesis and the fact that if $\ell = \exists(a_i : P_i \to P_{i+1}, c_{P_{i+1}})$ is taken as a hook, then the next context will be P_{i+1} and $f_{i+1} \circ a_i = f_i$.

Finally, we consider the case where ℓ is negative, $\ell = \neg\exists(a'_i : P_i \to Q_i, d_i)$, and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$. In this case, we have to prove that if there is a morphism $f : Q_i \to P$ satisfying $f \circ a'_i = f_i$, then f does not satisfy d_i. The proof makes use of the FinCol property (see Prop. 2) of Pattern that ensures that if there is such a morphism f there will be a P_j and a morphism $Q_i \to P_j$ that allow us to apply the partial lift rule. Moreover, that rule will be applied because if NT is saturated, T_j will be semi-saturated, which means that all possible partial lift rules will be applied to the hook in B_j. At this point, the Shifting Lemma, together with the inductive hypothesis, will ensure that f does not satisfy d_i.

Let us see now the detailed proof:

1. P is a weak ω-closed pattern: If $\langle n, L, n' \rangle \in \Rightarrow_P$ then, according to the construction of colimits in Prop. 2, there exists $\langle n_i, L, n'_i \rangle \in \Rightarrow_{P_i}$ with $f_i(n_i) = n$ and $f_i(n'_i) = n'$, for some $i \geq 0$. Moreover, since the tableau is saturated, the unfolding rule has been applied in some branch B_{j_1}, $j_1 \geq i$, in NB with context P_{j_1}, over the path expression $\langle a_{ij_1}(n_i), L, a_{ij_1}(n'_i) \rangle \in \Rightarrow_{P_{j_1}}$. That is, one literal $\exists(u, \texttt{true})$ with $u{:}P_{j_1} \to P'_{j_1} \in \texttt{Unfold}(P_{j_1}, \langle a_{ij_1}(n_i), L, a_{ij_1}(n'_i) \rangle)$ will be added to B_{j_1}. Since we assume that NT is saturated, there will be some k, $k \geq j_1$, where if $|Shift(\exists(u, \texttt{true}))| =$

$\exists(u', \texttt{true})$, and u' is not an isomorphism, it will be chosen as a hook at B_k. Then, according to Def. 10, the new context P_{k+1} will include either a relation $\langle a_{ik+1}(n_i), \{x\}, a_{ik+1}(n_i')\rangle$ or, both relations $\langle a_{ik+1}(n_i), \{x_1\}, a_{j_1k+1}(m_{j_1})\rangle$ and $\langle a_{j_1k+1}(m_{j_1}), L^{x_1}, a_{ik+1}(n_i')\rangle$, depending on which language is the one associated to the unfolding morphism u, that is, if either it is $\{x\} \in d_L$ or $x_1 \cdot L^{x_1} \in d_L$, respectively. In the first case, it means that the colimit P will include the edge expression $\langle n, \{x\}, f_{j_1}(m_{j_1})\rangle \in \rightarrow_{G_P}$, thus P satisfies to be a weak ω-closed pattern. In the second case, it means that the colimit P will include the following edge and path expression:

- $\langle n, \{x_1\}, f_{j_1}(m_{j_1})\rangle \in \rightarrow_{G_P}$, and
- $\langle f_{j_1}(m_{j_1}), L^{x_1}, n'\rangle \in \Rightarrow_P$

If u' is an isomorphism, we arrive to the same conclusion, since it would mean that, depending on which morphism is u,

- either $\langle a_{ik}(n_i), \{x\}, a_{ik}(n_i')\rangle$ or
- $\langle a_{ik}(n_i), \{x_1\}, a_{j_1k}(m_{j_1})\rangle$ and $\langle a_{j_1k}(m_{j_1}), L^{x_1}, a_{ik}(n_i')\rangle$

would already be in P_k and, as a consequence, either $\langle n, \{x\}, f_{j_1}(m_{j_1})\rangle \in \rightarrow_{G_P}$, or $\langle n, \{x_1\}, f_{j_1}(m_{j_1})\rangle \in \rightarrow_{G_P}$ and $\langle f_{j_1}(m_{j_1}), L^{x_1}, n'\rangle \in \Rightarrow_P$.
Again, by saturation of NT, at some further branch B_{j_2}, the unfolding rule will be applied to $\langle f_{j_1}(m_{j_1}), L^{x_1}, n'\rangle$, and so on. In the limit, one of the following situations should hold: i) either the unfolding rule is applied finitely until some finite path labelled with $s \in \Sigma^+$ is completely included in $\rightarrow^+_{G_P}$, for some $s \in L$; or ii) the unfolding is applied infinitely, including in P an infinite number of edges $n \xrightarrow{x_1} f_{j_1}(m_{j_1}) \xrightarrow{x_2} f_{j_2}(m_{j_2}), \ldots, f_{j_k}(m_{j_k}) \xrightarrow{x_{k+1}} f_{j_{k+1}}(m_{j_{k+1}}), \ldots$, which means that we have

$$\{\langle n, \{x_1\}, f_{j_1}(m_{j_1})\rangle, \ldots, \langle n, \{x_1x_2 \ldots x_{k+1}\}, f_{j_{k+1}}(m_{j_{k+1}})\rangle, \ldots\} \subseteq \rightarrow^+_{G_P}$$

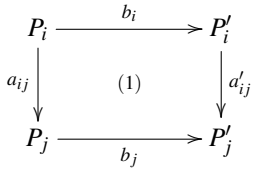
and $\{x_1,\ x_1x_2,\ \ldots,\ x_1x_2 \ldots x_{k+1},\ \ldots\} \subseteq \texttt{Pref}(s)$, with $s \in L$. In any case, we can conclude that P is a weak ω-closed pattern.

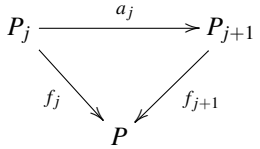
2. P satisfies c: We will show by induction on the nesting level of literals that for every literal ℓ in a branch B_i in NB, $f_i \models \ell$. In particular, this means that f_0 satisfies all the literals in the first branch implying, by Lemma 7, that P satisfies c.
The base case is direct, since $P \models \texttt{true}$.
For the general case, if ℓ is a literal of nesting level $n+1$ in some branch B_i, we will first consider the case where there is a j, with $j \geq i$, where $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level smaller than $n+1$ and for every k, with $i \leq k < j$, $\texttt{Shift}(a_{ik}, \ell)$ has not been chosen as the hook of B_k. Otherwise, we consider two cases, depending on whether ℓ is positive or negative.

- Let us assume without loss of generality that j is the smallest index where $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level smaller than $n+1$, i.e., $|\texttt{Shift}(a_{ij-1}, \ell)|$ has nesting level $n+1$ and $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level n. Since the nested tableau is saturated, $|\texttt{Shift}(a_{ij}, \ell)|$, will be part of the inner condition of the opener of T_j and, hence, it will be part of the condition c_{P_j} for that tableau. Moreover, since the nesting level of $|\texttt{Shift}(a_{ij}, \ell)|$ is at most n, we may

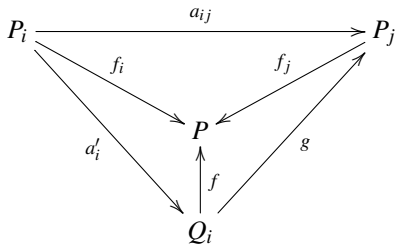
assume, by inductive hypothesis, that $f_j \models \ell'$ for all literals ℓ' in B_j that are in $|\texttt{Shift}(a_{ij}, \ell)|$, and by Lemma 7 $f_j \models |\texttt{Shift}(a_{ij}, \ell)|$. Finally, by Lemma 1, $f_j \circ a_{ij} \models \ell$, i.e., $f_i \models \ell$.

- If $\ell = \exists(b_i : P_i \to P'_i, d_i)$ and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$, by saturation, there must exist a j, with $j \geq i$, where $|\texttt{Shift}(a_{ij}, \exists(b_i : P_i \to P'_i, d_i))|$ is taken as the hook of B_j. Moreover, we may assume that if $\texttt{Shift}(a_{ij}, \exists(b_i, d_i)) = \exists(b_j : P_j \to P'_j, \texttt{Shift}(a'_{ij}, d_i))$, with P'_j and a'_{ij} defined by the pushout diagram (1) below, then $|\texttt{Shift}(a_{ij}, \exists(b_i, d_i))| = \exists(b_j : P_j \to P'_j, |\texttt{Shift}(a'_{ij}, d_i)|)$. The reason is that if b_j would be an isomorphism then $|\texttt{Shift}(a_{ij}, \ell)|$ would have nesting level smaller than $n+1$.
  Now, if $\exists(b_j : P_j \to P'_j, |\texttt{Shift}(a'_{ij}, d_i)|)$ is the hook for B_j, this means that $P'_j = P_{j+1}$, $b_j = a_j$ and the diagram below on the right commutes, since it is part of the cocone defining P. And it also means that $|\texttt{Shift}(a'_{ij}, d_i)|$ is part of the condition for T_{j+1}. Moreover, since ℓ has nesting level $n+1$, we may assume, by inductive hypothesis, that $f_{j+1} \models \ell'$ for all literals ℓ' in B_{j+1} that are in $|\texttt{Shift}(a'_{ij}, d_i)|$, and by Lemma 7 $f_{j+1} \models |\texttt{Shift}(a'_{ij}, d_i)|$. So, $f_j \models |\texttt{Shift}(a_{ij}, \exists(b_i, d_i))|$. Finally, by Lemma 1, this implies that $f_j \circ a_{ij} \models \ell$, but $f_j \circ a_{ij} = f_i$, thus $f_i \models \ell$.

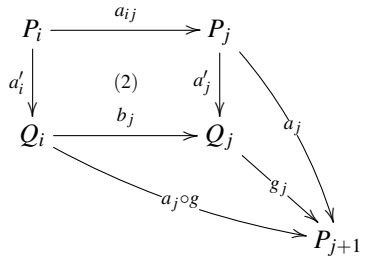




- If $\ell = \neg\exists(a'_i : P_i \to Q_i, d_i)$ and, for every $j > i$, $|\texttt{Shift}(a_{ij}, \ell)|$ has nesting level $n+1$, to prove that $f_i \models \ell$, we need to show that there does not exist a morphism $f : Q_i \to P$ such that $f \circ a'_i = f_i$ and $f \models d_i$. Hence, if there is no morphism $f : Q_i \to P$, then $f_i \models \ell$. Thus, let us assume that this morphism $f : Q_i \to P$ exists and let us show that f does not satisfy d_i.
  Following Prop. 2, there must exist P_j, with $j \geq i$, and a morphism $g : Q_i \to P_j$ with $g \circ a'_i = a_{ij}$ and $f = f_j \circ g$:



  Let $\exists(a_j : P_j \to P_{j+1}, c_j)$ be the hook in B_j, for some condition c_j. By the universal property of pushout diagram (2) below, since $a_j \circ a_{ij} = a_j \circ g \circ a'_i$, there must exist a morphism $g_j : Q_j \to P_{j+1}$ such that $a_j \circ g = g_j \circ b_j$ and $a_j = g_j \circ a'_j$.



So, we have the following situation:

- $\exists(a_j : P_j \to P_{j+1}, c_j)$ is the hook in B_j, for some condition c_j.
- $|\texttt{Shift}(a_{ij}, \neg\exists(a'_i, d_i))| = \neg\exists(a'_j : P_j \to Q_j, |\texttt{Shift}(b_j, d_i)|)$ is a negative literal in B_j. Otherwise, a'_j would be an isomorphism and $|\texttt{Shift}(a_{ij}, \ell)|$ would have nesting level smaller than $n+1$.
- There is a morphism $g_j : Q_j \to P_{j+1}$ such that $a_j = g_j \circ a'_j$.

This means that we can apply a partial lift rule to these literals. Moreover, since we assume that the tableau NT is saturated, the partial lift rule had to be applied in that branch B_j, and the condition $|\neg\texttt{Shift}(g_j, \texttt{Shift}(b_j, d_i))|$ (which is equal to $|\neg\texttt{Shift}(g_j \circ b_j, d_i)|$) will be part of the inner condition of the opener and, therefore, of the condition for T_{j+1}. Since the nesting level of $|\neg\texttt{Shift}(a_j \circ g, d_i)|$ is at most n, by inductive hypothesis, $f_{j+1} \models \ell'$ for every ℓ' in B_{j+1} and in $|\neg\texttt{Shift}(a_j \circ g, d_i)|$. Then, by Lemma 7, $f_{j+1} \models |\neg\texttt{Shift}(a_j \circ g, d_i)|$, and by Lemma 1, we obtain $f \models \neg d_i$ since $f = f_j \circ g = f_{j+1} \circ a_j \circ g$. ■

Appendix B. Example of a nested tableau refutation

In this appendix we show a nested tableau refutation for condition $c = c_1 \wedge c_2 \wedge c_3$ with $c_1 = \exists(\emptyset \longrightarrow [\bullet \overset{x^+}{\Rightarrow} \bullet], \texttt{true})$, $c_2 = \exists(\emptyset \longrightarrow [\bullet \overset{y^+}{\Rightarrow} \bullet], \texttt{true})$, and $c_3 = \neg\exists(\emptyset \longrightarrow \left[\begin{smallmatrix} \bullet \overset{x}{\rightarrow} \bullet \\ \bullet \underset{y}{\rightarrow} \bullet \end{smallmatrix}\right], \texttt{true})$.

The nested tableau consists of 6 simple tableaux, depicted in Figures B.11, B.12 and B.13.

In Fig. B.11, we start bulding a tableau T_0 with `true` and adding the literals of condition c by means of the extension rule (*ext*). Then we choose as hook the positive literal c_1 and apply twice the lift rule (L): first to c_1 and c_2, obtaining c_4, and then to c_4 and c_3 obtaining c_5. T_0 is a semi-saturated tableau with a leaf c_5 which is equivalent to the initial condition c. Although it is not necessary for semi-saturation, note that in this tableau it is not possible to apply the split reduction rule (since the morphism in the only negative literal c_3 is not split mono) nor the unfolding rule (since $\emptyset$ context has no path expressions).

Now $c_5 = \exists(\emptyset \longrightarrow [\bullet \overset{x^+}{\Rightarrow} \bullet], c_6 \wedge c_7)$ is an opener, so that we open a new tableau T_1 of context $[\bullet \overset{x^+}{\Rightarrow} \bullet]$ for its inner condition $c'_1 = c_6 \wedge c_7$. Since there exists a path expression $\langle n, x^+, n' \rangle$ in the context of T_1, it is possible to apply the unfolding rule to obtain the new literals c_8 and c_9 with morphisms u_0 and u_1 respectively. Then we proceed in each branch of T_1 as before, by taking c_8 and c_9 as hooks, respectively, and applying the lift rule (L) until obtaining two leaves, c_{14} and c_{16}, in this semi-saturated tableau T_1. These conditions are the openers for adding two more tableaux (in the nested tableau we are building) for their inner conditions c'_2 and c'_3 respectively.

In Fig. B.12, we can see a closed nested tableau for condition c'_2 built from tableaux T_2 and T_4. Note that in the tableau T_4 we have only a negative literal c'_4, but positive literals are added by means of the unfolding rule, since there exists a path expression $\langle n, y^+, n' \rangle$ in the context of T_4. Now, in both branches the partial lift rule (PL) can be applied to obtain the leaves c_{22} and c_{23} which are both equivalent to `false`. Similarly, in Fig. B.13 we show a closed nested tableau for condition c'_3.

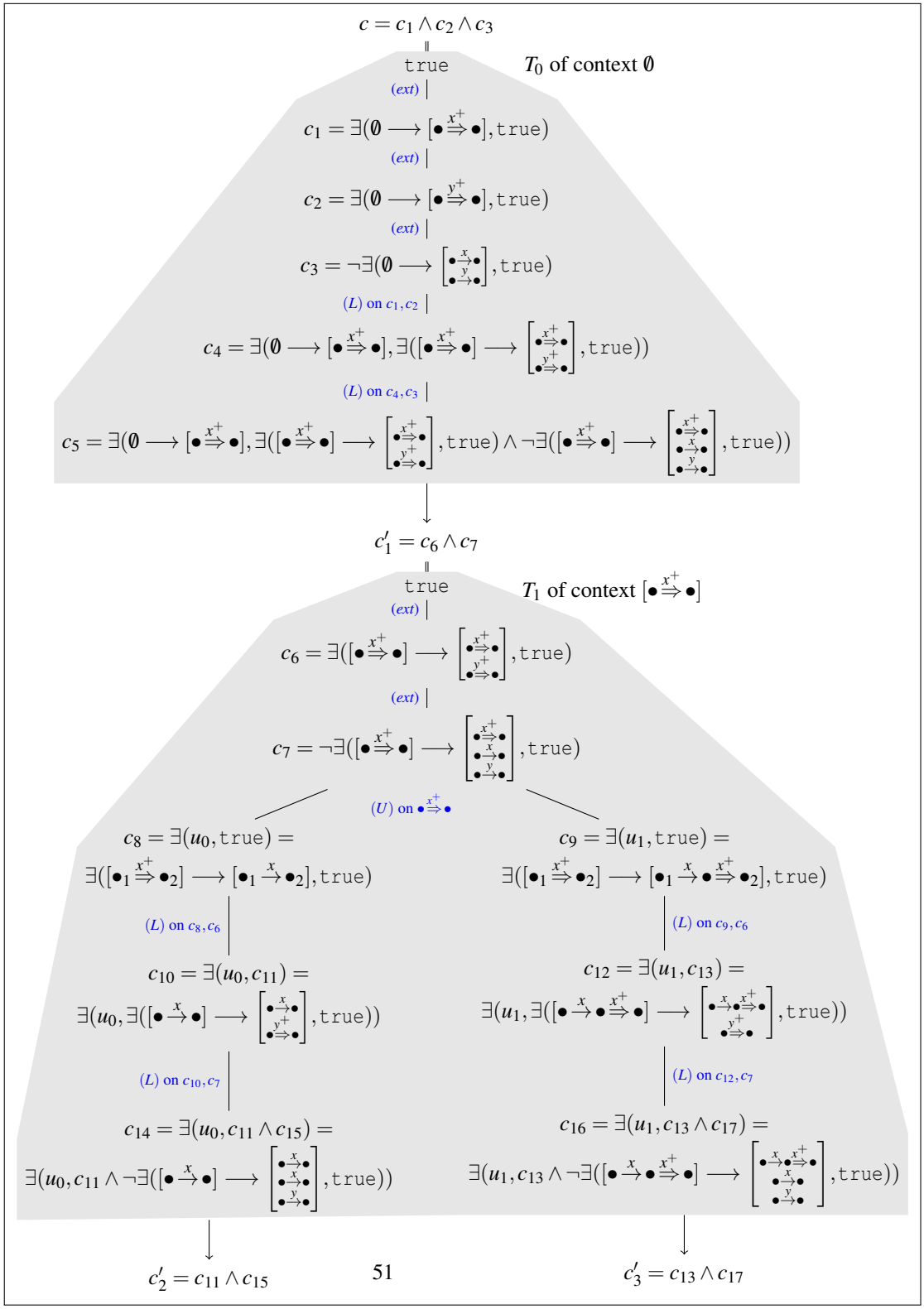


Figure B.11: Part of a nested tableau for condition c

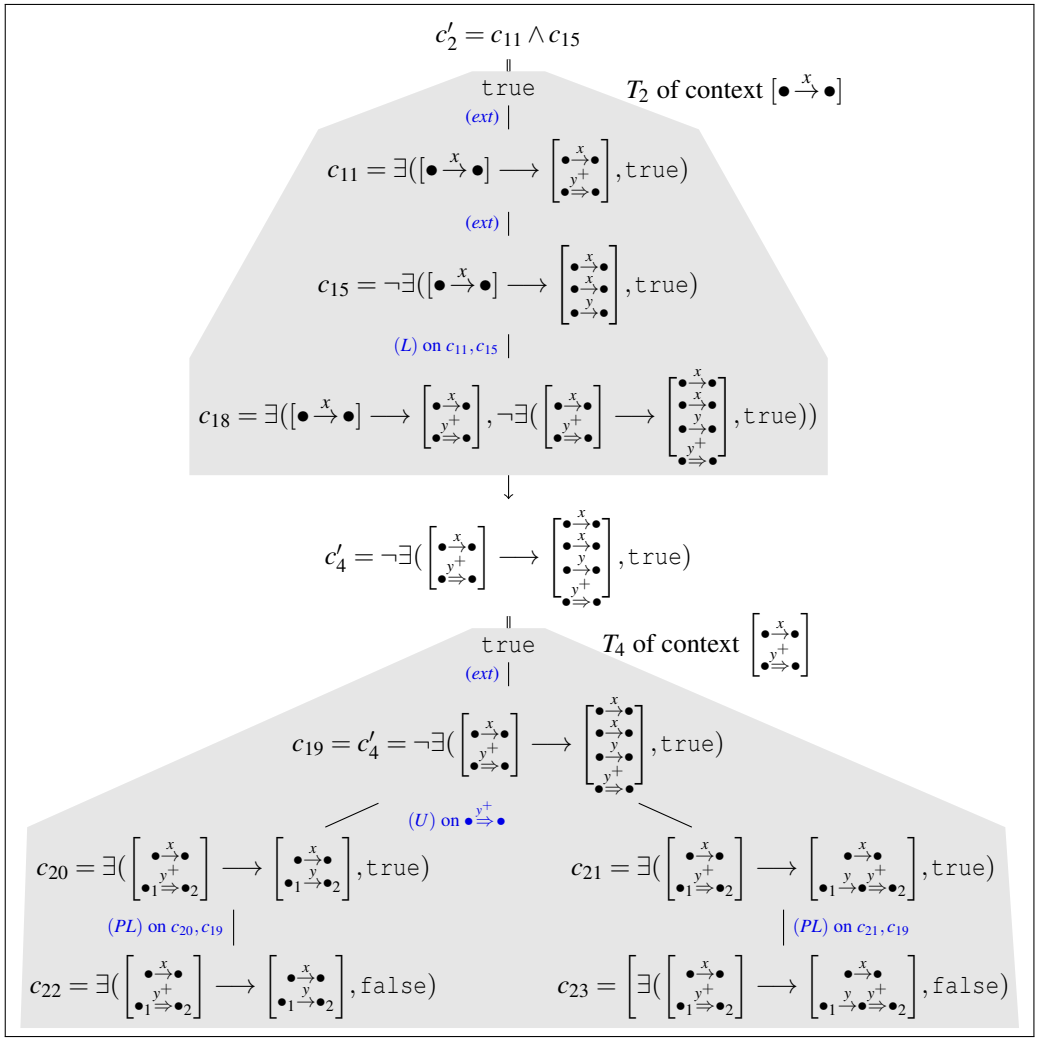


Figure B.12: Closed nested tableau for condition c_2'

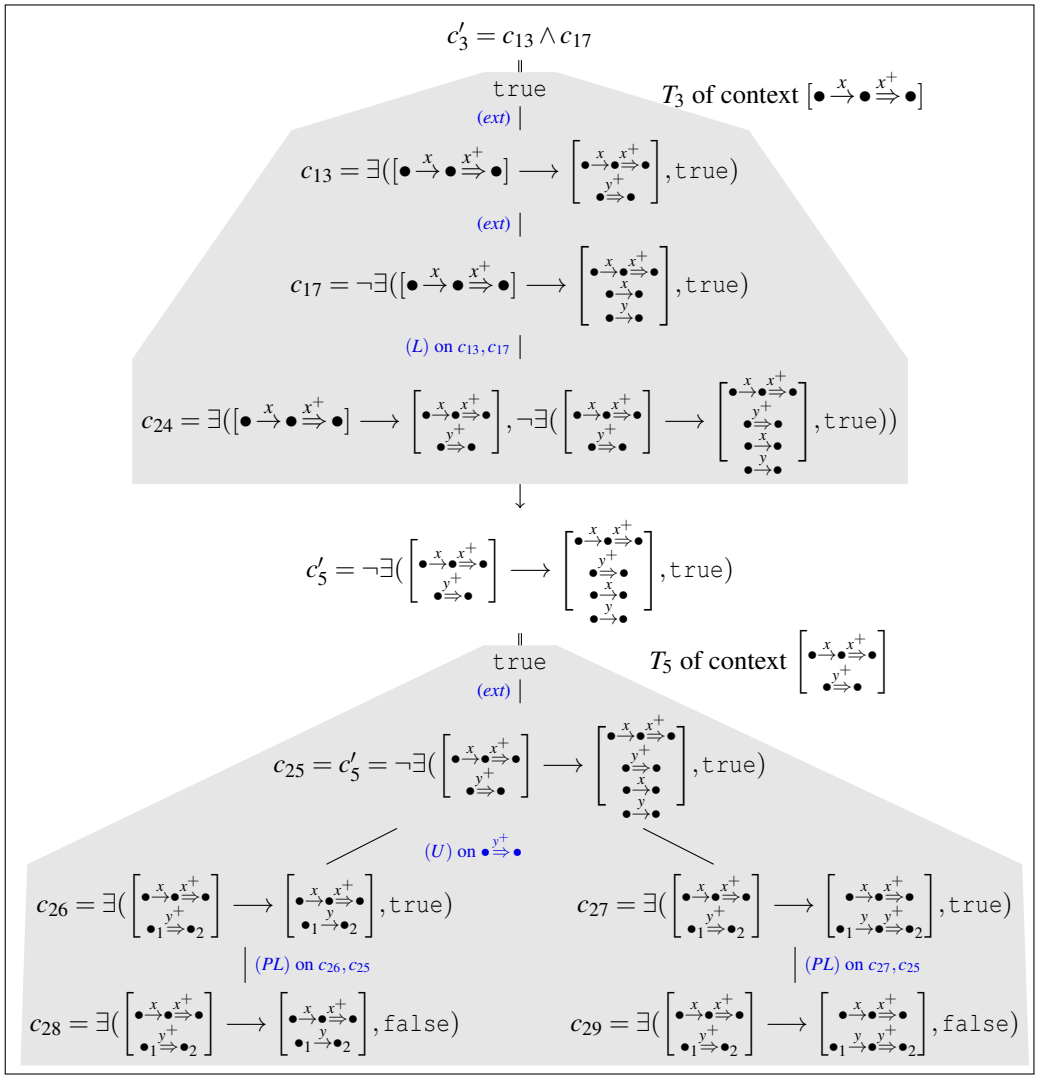


Figure B.13: Closed nested tableau for condition c'_3