

Reasoning with graph constraints

Fernando Orejas, Hartmut Ehrig, Ulrike Prange

▶ **To cite this version:**

Fernando Orejas, Hartmut Ehrig, Ulrike Prange. Reasoning with graph constraints. Formal Aspects of Computing, 2009, 22 (3), pp.385-422. 10.1007/s00165-009-0116-9 . hal-00497320

HAL Id: hal-00497320
https://hal.science/hal-00497320

Submitted on 4 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Under consideration for publication in Formal Aspects of Computing

Reasoning with Graph Constraints

Fernando Orejas[1], Hartmut Ehrig[2], and Ulrike Prange[2]

[1]Dpto de L.S.I., Universitat Politècnica de Catalunya, Campus Nord, Mòdul Omega, Jordi Girona 1-3, 08034 Barcelona, Spain.
[2]Fak. IV, Technische Universität Berlin, Franklinstrasse 28/29, 10587 Berlin, Germany.

Abstract. Graph constraints were introduced in the area of graph transformation, in connection with the notion of (negative) application conditions, as a form to limit the applicability of transformation rules. However, we believe that graph constraints may also play a significant role in the area of visual software modelling or in the specification and verification of semi-structured documents or websites (i.e. HTML or XML sets of documents). In this sense, after some discussion on these application areas, we concentrate on the problem of how to prove the consistency of specifications based on this kind of constraints. In particular, we present proof rules for two classes of graph constraints and show that our proof rules are sound and (refutationally) complete for each class. In addition we study clause subsumption in this context as a form to speed up refutation.

Keywords: Graph constraints, Visual modelling, Graph transformation

1. Introduction

Graph constraints were introduced in the area of graph transformation, together with the notion of (negative) application conditions, as a form to limit the applicability of transformation rules [EhH86, HHT96, HeW95, EEHP04, HaP05, HaP06]. More precisely, a graph constraint is the graphical description of some kind of pattern that must be present (or must not be present) in the graphs that we are transforming. In particular, a transformation would be illegal if the resulting graph would violate any of the given constraints. Graph constraints have been studied mainly in connection with negative application conditions. These conditions are constraints that are associated to the left-hand side or the right-hand side of a graph transformation rule. Then, one such rule would be applicable to a given graph if the left-hand side application conditions are satisfied by the given graph (or rather by the rule matching) and the right-hand side application conditions are satisfied by the result of the transformation (or rather by its comatch). In this context, most of the above-mentioned work is related to the extension of basic graph transformation concepts and results to the use of application conditions and constraints, and to show how one can transform a set of constraints into application conditions for the given transformation rules. Other work related to these notions has studied the detection of conflicts for graph transformation with application conditions [LEO06], or the expressive power of some kinds of graph constraints [Ren04].

We believe that graph constraints can go beyond their use in connection to graph transformation. More precisely,

Correspondence and offprint requests to: Fernando Orejas, Dpto de L.S.I., Universitat Politècnica de Catalunya, Campus Nord, Mòdul Omega, Jordi Girona 1-3, 08034 Barcelona, Spain e-mail: orejas@lsi.upc.edu

there are two areas in which we think that graph constraints may play an interesting role. The first one is the area of visual software modelling. The second one is the specification and verification of classes of semi-structured documents, including the specification and verification of websites (i.e. HTML or XML sets of documents).

In the area of visual modelling, especially in the context of UML modelling, models are designed using different kinds of diagrams. However, if we have to impose some specific constraints on the models, then we have to use a textual notation as OCL. We consider that this situation is quite inconvenient. Especially, when we want to express constraints on the structure of the model, we think that using a graphical notation which is close to the visual description of the model is much more clear and intuitive than using some textual expression where one has to previously code or represent that structure.

Some very recent work that is related to this kind of application of graph constraints is the work by de Lara and Guerra on the specification and synthesis of model transformations [LaG08]. In that paper, they describe model transformations using graph constraints over triple graphs. Then, these constraints are transformed into (triple) graph grammar rules that can be used to implement the model transformation specified by the constraints. As a first step for the synthesis of the graph rules, some inference steps are done using deduction rules which are similar to some of the rules that are used in this paper.

Other work that, in a sense, is related to the use of graph constraints in visual modeling is the work by Parisi and Koch on the specification and analysis of access control policies (see, e.g. [KMP05]). In particular, they specify access control policies using graph constraints to describe the valid states of a system, and graph transformation rules to specify operations. Interestingly, they use some form of deduction on constraints to check the consistency of a policy. Unfortunately, the kind of deduction used may be considered quite ad-hoc and incomplete.

On the other hand, we know two kinds of approaches for the specification and verification of semi-structured documents. The first one [AlF06, EEFN03] is based on extending a fragment of first-order logic allowing us to refer to the components of the given class of documents (in particular, using XPath notation). This approach, in our opinion, poses two kinds of problems. On one hand, from a technical point of view, the extension of first-order logic to represent XML patterns has to make use of associative-commutative operators. This may make deduction difficult to implement efficiently, since using unification in inference rules may be very costly (in general, two arbitrary atoms may have a doubly exponential amount of most general unifiers). As a consequence, the approaches presented in [AlF06, EEFN03] present specification languages that allow us to specify classes of documents, and tools that allow us to check if a given document (or a set of documents) follows a specification. However, they do not consider the problem of defining deductive tools to analyze specifications, for instance for looking for inconsistencies. On the other hand, from a pragmatic point of view, this kind of specifications can be quite verbose and this may make the resulting specifications unpleasant to read and to write.

The other approach that we know [Jel00], which we consider especially interesting, has a more practical nature. Schematron is a language and a tool that is part of an ISO standard (DSDL: Document Schema Description Languages). The language allows us to specify constraints on XML documents by describing directly XML patterns (using XML) and expressing properties about these patterns. Then, the tool allows us to check if a given XML document satisfies these constraints. However, we consider that there are two problems with this approach. The most important one is that this work lacks proper foundations. The other one is that the kind of patterns that can be expressed in the Schematron language could be a bit limited. On the other hand, as in the approaches mentioned above, Schematron provides no deductive capabilities.

In this paper we start the study of graph constraints as a specification formalism. In particular, we study their underlying logic, providing inference rules that would allow us to prove the consistency (or satisfiability) of specifications. Actually, we show that these rules are sound and refutationally complete for the class of constraints considered. It must be noted that, as it is well-known, the fact that our inference rules are refutationally complete means that we have a complete method to prove consequences of our specifications. In particular, if we want to check if a given property is a consequence of a specification then it is enough to see if the given specification, together with the negation of the property, is inconsistent.

Some very recent work that is very related to ours is [Pen08]. In that paper, Pennemann proposes a proof system for nested graph constraints, a generalization of the kind of constraints considered in our work. The proof system is proven sound but not complete. In addition, Pennemann describes an implementation of his approach providing interesting results.

It must also be noted that the results that we present are quite more general than what they actually may seem. Following recent work on algebraic graph transformation (see, e.g., cite [EEPT06]), our results apply not only to plain graphs, but generalize to a large class of structures including typed and attributed graphs (we discuss this issue in more detail in the conclusion). In particular, instead of a logic of *graph* constraints we could speak of a logic of *pattern* constraints, since our results would also apply to reasoning about constraints based on other kinds of patterns,

like XML patterns. In this sense, we consider that the work that we present in this paper provides the basis for defining the logical foundations of Schematron, and for extending it with more powerful constraints and with deduction capabilities. In particular, the XML patterns that are used in Schematron can be seen just as the textual representation (or, rather, the XML representation) of a subclass of the graph constraints that we consider. In particular, our work could be used to provide deductive capabilities to analyze the consistency of Schematron specifications.

The work that we present is not the first logic to reason about graphs. With different aims, in a series of papers (for a survey, see [Cou97]) Courcelle has studied in detail the definition and use of a graph logic (in the following called CL, from Courcelle Logic). His approach can be seen as a coding of graphs and graph properties into first-order or monadic second-order logic. In particular, the approach is based on the use of some predicates describing the existence of nodes and edges which, together with some given axioms, provide an axiomatization of the basic graph theory. Then, one can express graph properties using standard first-order or monadic second-order formulas over these predicates. Our constraints can be seen as a fragment of CL in the sense that a graph constraint can be coded into a sentence in that logic. Actually, nested constraints have been proved equivalent to the first-order fragment of CL [HaP08]. As a consequence, there are two main issues that one may consider. On one hand, whether graphs constraints, as advocated in this paper, are useful as a modeling formalism. On the other hand, we can question whether it is really needed to develop proof techniques for our constraints, since we can do this indirectly: by coding the constraints into CL and using standard logic deduction. In particular, with respect to the first issue, we could think of directly using CL to write our specifications. However, we think that for modeling and specification purposes, graph constraints provide a much more friendly and intuitive formalism than CL. With respect to the second issue, we think that there are two main reasons that justify our work in this direction. First, studying directly the constraints logic gives you insights about the logic that we would not obtain using the coding. For instance, our completeness proofs implicitly tell us how we can design procedures to build models for a given set of constraints. This is interesting for applications like the one presented in [LaG08], where building a model is, in a sense, equivalent to synthesizing the specified model transformation. And, second, we believe that we can gain significant efficiency. Actually, this kind of discussion is not new. For instance, the development of proof techniques for first-order logic with equality has sometimes been questioned, considering that one could use the standard techniques for first-order logic without equality together with an axiomatization of the equality predicate. However, the study of first-order logic with equality has allowed the development of powerful techniques which are the basis of very efficient tools. In this sense, in [Pen08] Pennemann compares his implementation for his proof system for nested constraints with an implementation based on coding the constraints into CL and then using some standard provers like VAMPIRE, DARWIN and PROVER9. The result is that his implementation outperforms the coding approach. Actually, in most examples considered, the above provers were unable to terminate in the given time (1 hour of cpu time). Unfortunately, these results can not be considered technically valid, since the completeness of Pennemann's proof system is not shown. In [BCKL06] CL is extended with temporal operators. In this case, the intention is to present a logic that can be used for the verification of graph transformation systems.This logic goes far beyond our aims.

This paper is organized as follows. In the next section we present the kind of graph constraints that we consider in this paper and some basic notions concerning refutation procedures. Moreover, we present a small example to motivate their use in connection with visual modeling. This example will be used as a running example in the rest of the paper. The following two sections are the core of the paper. They present inference rules for two classes of graph constraints showing, in both cases, their soundness and completeness. Then, in Section 5, we present some techniques that may be used to speed up refutation procedures. In particular, we present a notion of subsumption, proving that subsumed clauses can be eliminated without losing completeness. Finally, in the conclusion we discuss some issues concerning the results that we present, in particular, their generality and the possible implementation of a deductive tool.

This paper extends and generalizes the work presented in [OEP08] in several ways. In particular, in addition to providing detailed proofs for all our results, the paper considers the general case where specifications are assumed to consist of arbitrary clauses, while in [OEP08] the specifications were assumed to be just sets of literals. In addition, this paper includes a new section about subsumption and clause elimination which was not present in [OEP08].

2. Graphs and Graph Constraints

In this section we present the basic notions that are used in this paper. First we present some notation and terminology needed. Then, in the second subsection we introduce the kind of graph constraints that we consider. Finally, in the third subsection, we introduce some standard basic concepts about refutation procedures For simplicity, we present our definitions in terms of plain directed graphs, although in some examples, for motivation, we deal with typed

attributed graphs. Anyhow, following the approach used in [EEPT06], it should not be difficult to show that our results generalize to a large class of (graphical) structures. In Section 6 we discuss this issue in more detail.

2.1. Graphs

As said above, all our notions and results will be presented in terms of plain directed graphs, i.e.:

Definition 1 ***(Graphs)*** *A* graph $G = (G^V, G^E, s^G, t^G)$ *consists of a set G^V of nodes, a set G^E of edges, a source function $s^G : G^E \to G^V$, and a target function $t^G : G^E \to G^V$.*

It may be noted that we do not explicitly state that the sets of nodes and edges of a graph are finite sets. That is, according to our definition, unless it is explicitly stated, graphs may be infinite. This issue is discussed in some detail in Sections 3 and 4.

All over the paper we will have to express that a certain graph G_1 is included into another graph G_2. Obviously, we could have done this through a subgraph relationship. However, G_2 may include several instances of G_1. For this reason, in order to be precise when specifying the specific instance in which we may be interested, we will deal with these inclusions using the notion of graph monomorphism:

Definition 2 ***(Graph morphisms)*** *Given the graphs $G = (G^V, G^E, s^G, t^G)$ and $H = (H^V, H^E, s^H, t^H)$, a* graph morphism *$f : G \to H$ is a pair of mappings, $f^V : G^V \to H^V$, $f^E : G^E \to H^E$ such that f commutes with the source and target functions, i.e. the diagrams below are commutative.*

$$\begin{array}{ccc} G^E & \xrightarrow[s^G]{} & G^V \\ \downarrow{\scriptstyle f^E} & & \downarrow{\scriptstyle f^V} \\ H^E & \xrightarrow[s^H]{} & H^V \end{array} \qquad \begin{array}{ccc} G^E & \xrightarrow[t^G]{} & H^V \\ \downarrow{\scriptstyle f^E} & & \downarrow{\scriptstyle f^V} \\ H^E & \xrightarrow[t^H]{} & G'^V \end{array}$$

A graph morphism $f : G \to H$ is a monomorphism *if f^V and f^E are injective mappings.*

In several results of the paper, given two graphs G, G' we will need to overlap them in all possible ways. This will be done using the construction $G \otimes G'$:

Definition 3 ***(Jointly surjective morphisms)*** *Two graph morphisms $m : H \to G$ and $m' : H' \to G$ are* jointly surjective *if $m^V(H^V) \cup m'^V(H'^V) = G^V$ and $m^E(H^E) \cup m'^E(H'^E) = G^E$.*

Given two graphs G and G', the set of all pairs of jointly surjective monomorphisms from G and G' is denoted by $G \otimes G'$, that is:

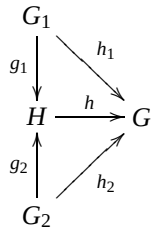
$$G \otimes G' = \{m : G \to H \leftarrow G' : m' \mid \text{ m and m}' \text{ are jointly surjective monomorphisms}\}.$$

The definition of $G \otimes G'$ in terms of sets of pairs of monomorphisms may look a bit more complex than needed but, as in the case of the inclusions, we often need to identify the specific instances of G and G' inside H. However, from an intuitive point of view, it is enough to consider that $G \otimes G'$ is the set of all graphs that can be seen as the result of overlapping G and G'.

Note that if G and G' are finite graphs then $G \otimes G'$ is also a finite set (up to isomorphism). This is needed because in several inference rules (see Sections 3 and 4) the result is a clause involving a disjunction related to a set of this kind. In particular, if $G \otimes G'$ is infinite so would be the corresponding disjunction. A property satisfied by graphs, which we use in the proofs of most results, is pair factorization:

Proposition 1 ***(Pair factorization)*** *Given two graph morphisms, $h_1 : G_1 \to G \leftarrow G_2 : h_2$, with the same codomain G there exists a graph H and morphisms $g_1 : G_1 \to H \leftarrow G_2 : g_2$ and $h : H \to G$ such that g_1 and g_2 are jointly surjective*

and the diagram below commutes:



Moreover, if h_1 and h_2 are monomorphisms so are g_1 and g_2.

Proof. We define the graph H as follows:

- $H^V = \{v \in G^V \mid \exists v_1 \in G_1^V \;\; h_1^V(v_1) = v\} \cup \{v \in G^V \mid \exists v_2 \in G_2^V \;\; h_2^V(v_2) = v\}$
- $H^E = \{e \in G^E \mid \exists e_1 \in G_1^E \;\; h_1^E(e_1) = e\} \cup \{e \in G^E \mid \exists e_2 \in G_2^E \;\; h_2^E(e_2) = e\}$
- For every $e \in H^E$, $s^H(e) = s^G(e)$ and $t^H(e) = t^G(e)$

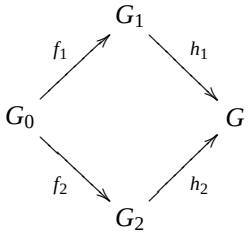
and we define g_1 and g_2 as follows:

- For every $v_1 \in G_1^V$, $g_1^V(v_1) = h_1^V(v_1)$ and for every $e_1 \in G_1^E$, $g_1^E(e_1) = h_1^E(e_1)$
- For every $v_2 \in G_2^V$, $g_2^V(v_2) = h_2^V(v_2)$ and for every $e_2 \in G_2^E$, $g_2^E(e_2) = h_1^E(e_2)$

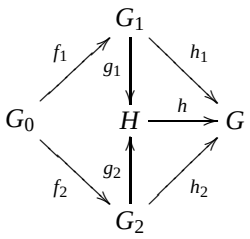
Now, by definition, g_1 and g_2 are jointly surjective and H is a subgraph of G. Let us call h the monomorphism associated to this inclusion. Moreover, notice that by definition if h_1 and h_2 are monomorphisms so are g_1 and g_2. We only have to prove that the diagram above commutes. But this is also a straightforward consequence of the definitions of H, g_1, g_2, and h. □

Extended pair factorization, which seems a generalization of pair factorization, is also used in our proofs. However we can see that extended pair factorization is really a straightforward consequence of pair factorization:

Proposition 2 ***(Extended pair factorization)*** *Given the commuting diagram below,*



there exist a graph H and morphisms $g_1 : G_1 \to H \leftarrow G_2 : g_2$, and $h : H \to G$ such that g_1 and g_2 are jointly surjective and the diagram below commutes:



Moreover, if h_1 and h_2 are monomorphisms so are g_1 and g_2

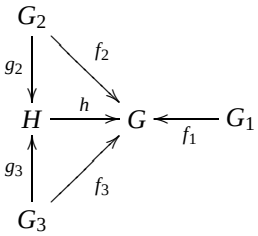
Proof. Let us define H, g_1, g_2, and h using pair factorization. Then we only need to prove that $g_1 \circ f_1 = g_2 \circ f_2$. Now, according to pair factorization we know that $h \circ g_1 \circ f_1 = h_1 \circ f_1 = h_2 \circ f_2 = h \circ g_2 \circ f_2$. But we know that h is a monomorphism, therefore $g_1 \circ f_1 = g_2 \circ f_2$. □

We may see that $\otimes$ is, in some sense, associative and commutative:

Proposition 3 *Given three graphs G_1, G_2 and G_3 then:*
$\{G \mid \langle f, G, g\rangle \in G_1 \otimes (G_2 \otimes G_3)\} = \{G \mid \langle f, G, g\rangle \in (G_2 \otimes G_1) \otimes G_3\} =$
$\{G \mid$ *there are jointly surjective monomorphisms* $f : G_1 \to G, g : G_2 \to G, h : G_3 \to G\}$

Proof. We start proving that if G is in $\{G \mid \langle f, G, g\rangle \in G_1 \otimes (G_2 \otimes G_3)\}$ then there are jointly surjective monomorphisms $f_1 : G_1 \to G$, $f_2 : G_2 \to G$, $f_3 : G_3 \to G$. Suppose G is in $\{G \mid \langle f, G, g\rangle \in G_1 \otimes (G_2 \otimes G_3)\}$, this means that there is a graph H and morphisms $f' : G_2 \to H \leftarrow G_3 : g'$ and $f : G_1 \to G \leftarrow H : g$ such that f' and g' are jointly surjective and so are f and g. But, then, it is routine to show that $f_1 = f : G_1 \to G$, $f_2 = g \circ f' : G_2 \to G$, and $f_3 = g \circ g' : G_3 \to G$ are jointly surjective.

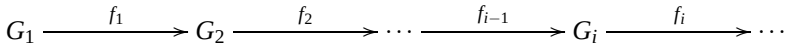
Now, we prove the converse inclusion. Suppose that there are jointly surjective monomorphisms $f_1 : G_1 \to G$, $f_2 : G_2 \to G$, $f_3 : G_3 \to G$. Using the pair factorization property there exist a graph H and monomorphisms g_2, g_3, and h:



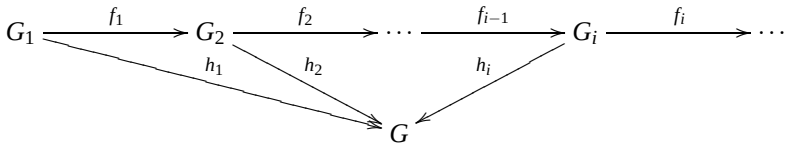
such that the diagram above commutes and g_2 and g_3 are jointly surjective. But this means that $\langle g_2, H, g_3\rangle \in G_2 \otimes G_3$. On the other hand, it is routine to prove that f_1 and h are jointly surjective, which means that $\langle f_1, G, m\rangle \in G_1 \otimes (G_2 \otimes G_3)$. The prove that $\{G \mid$ there are jointly surjective monomorphisms $f : G_1 \to G, g : G_2 \to G, h : G_3 \to G\} = \{G \mid \langle f, G, g\rangle \in (G_2 \otimes G_1) \otimes G_3\}$ is similar. □

Finally, the last property that we need for our results, which is also satisfied by graphs, is the existence of infinite colimits (satisfying an additional minimality property) for sequences of monomorphisms. Intuitively, these colimits are the union of the graphs in the sequence. Actually, in the category of graphs we have colimits for arbitrary diagrams. To be more precise:

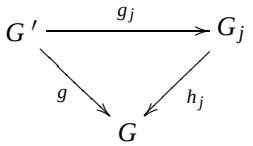
Proposition 4 ***(Infinite colimits)*** *Given a sequence of monomorphisms:*



there exists a colimit:



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



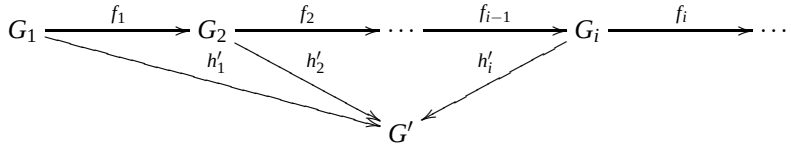
Proof. We define the graph G as follows:

- $G^V = (\bigcup_{1\leq i} G_i^V)/\equiv_V$, where $\equiv_V$ is the least equivalence relation satisfying that for every i and every $v \in G_i^V$ $v \equiv_V f_i(v)$.
- $G^E = (\bigcup_{1\leq i} G_i^E)/\equiv_E$, where $\equiv_E$ is the least equivalence relation satisfying that for every i and every $e \in G_i^E$ $e \equiv_E f_i(e)$.
- For every $e \in G_i^E$, $s^G(|e|) = |s^{G_i}(e)|$ and $t^G(|e|) = |t^{G_i}(e)|$.

Moreover, for every i we define the morphism $h_i : G_i \to G$ as follows:

- For every $v \in G_i^V$, $h_i^V(v) = |v|$.
- For every $e \in G_i^E$, $h_i^E(e) = |e|$.

Now, it should be obvious that, by definition, the graph G and the morphisms h_i are a cocone for the above diagram. We may see that it satisfies the universal property for colimits. Suppose that the diagram:



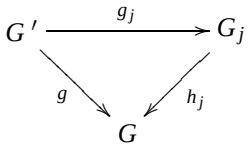
is also a cocone. We define the following morphism $h : G \to G'$:

- For every $v \in G_i^V$, $h^V(|v|) = h_i'^V(v)$.
- For every $e \in G_i^E$, $h^E(|e|) = h_i'^E(e)$.

By definition, for every i, $h'_i = h \circ h_i$. Now, suppose that $h' : G \to G'$ also satisfies that for every i, $h'_i = h' \circ h_i$ let us see that $h = h'$:

- For every $v \in G_i^V$, $h^V(|v|) = h_i'^V(v) = h'^V(h_i^V(v) = h'^V(|v|)$.
- For every $e \in G_i^E$, $h^E(|e|) = h_i'^E(e) = h'^E(h_i^E(e) = h'^E(|e|)$.

Therefore we have proved that G together with the morphisms h_i are a colimit for the diagram above. Let us now prove that this colimit satisfies that for every monomorphism $g : G' \to G$, such that G' is a finite graph, there is a j and a monomorphism $g_j : G' \to G_j$ such that the diagram below commutes:



Let G' be a finite graph and suppose that $g : G' \to G$. For each i, let $G'_i \subseteq G$ be the image of G_i by h_i. It should be noted that, by definition, we have that, for each i, $G'_i \subseteq G'_{i+1}$ and, moreover, G_i and G'_i are isomorphic, since the morphisms h_i are injective. Let $g'_i : G'_i \to G_i$ be that isomorphism, for each i. In addition, we also have:

- $G^V = \bigcup_{1\leq i} G_i'^V$, and
- $G^E = \bigcup_{1\leq i} G_i'^E$

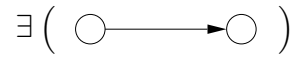
Now, since G' is finite, there must be a j such that for every $v \in G'^V$: $g^V(v) \in G_j'^V$ and for every $e \in G'^E$: $g^E(e) \in G_j'^E$. Then, we can define the required g_j as follows:

- For every $v \in G'^V$, $g_j^V(v) = g_j'^V(g^V(v)$.
- For every $e \in G'^E$, $g_j^E(e) = g_j'^E(g^E(e)$.

Then, by definition, g_j commutes the above diagram. □

2.2. Graph Constraints

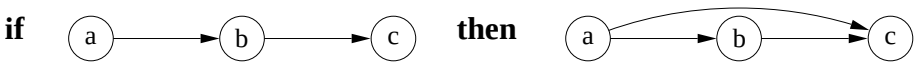
The underlying idea of a graph constraint is that it should specify that certain patterns must be present (or must not be present) in a given graph. For instance, the simplest kind of graph constraint, $\exists C$, specifies that a given graph G should include (a copy of) C. For instance, the constraint:



specifies that a graph should include at least one edge. Obviously, $\neg\exists C$ specifies that a given graph G should not include (a copy of) C. For instance, the constraint:



specifies that a given graph G should not include two different edges between any two nodes. A slightly more complex kind of graph constraints are atomic constraints of the form $\forall(c : X \to C)$ where c is a monomorphism (or, just, an inclusion). This constraint specifies that whenever a graph G includes (a copy of) the graph X it should also include (a copy of) its extension C. However, in order to enhance readability (the monomorphism arrow may be confused with the edges of the graphs), in our examples we will display these kinds of constraints using an **if - then** notation, where the two graphs involved have been labelled to implicitly represent the given monomorphism. For instance, the constraint:



specifies that a graph must be transitive, i.e. the constraint says that for every three nodes a, b, c if there is an edge from a to b and an edge from b to c then there should be an edge from a to c.

Obviously, graph constraints can be combined using the standard connectives $\vee$ and $\neg$ (as usual, $\wedge$ can be considered a derived operation). In addition, in [EEHP04, Ren04] a more complex kind of constraints, namely nested constraints, is defined, but we do not consider them in this paper.

Definition 4 ***(Syntax of graph constraints)*** *An* atomic graph constraint $\forall(c : X \to C)$ *is a graph monomorphism $c : X \to C$, where X and C are finite graphs. An atomic graph constraint $\forall(c : X \to C)$, where $X = \emptyset$, is called a basic atomic constraint (or just a basic constraint) and will be denoted $\exists C$.*

Graph constraints *are logic formulas defined inductively as usual:*

- *Every atomic graph constraint is a graph constraint.*
- *If α is a graph constraint then $\neg\alpha$ is also a graph constraint.*
- *If α_1 and α_2 are graph constraints then $\alpha_1 \vee \alpha_2$ is also a graph constraint.*

Satisfaction of constraints is also defined inductively following the intuitions described above.

Definition 5 ***(Satisfaction of graph constraints)*** *A graph G satisfies a constraint α, denoted $G \models \alpha$ if the following holds:*

- *$G \models \forall(c : X \to C)$ if for every monomorphism $h : X \to G$ there is a monomorphism $f : C \to G$ such that $h = f \circ c$.*
- *$G \models \neg\alpha$ if G does not satisfy α.*
- *$G \models \alpha_1 \vee \alpha_2$ if $G \models \alpha_1$ or $G \models \alpha_2$.*

It may be noted that, according to these definitions, the constraint $\exists\emptyset$, where $\emptyset$ denotes the empty graph, is satisfied by any graph, i.e. $\exists\emptyset$ may be considered the trivial *true* constraint.

We assume that our specifications consist of clauses of the form $L_1 \vee \cdots \vee L_n$, where each *literal* L_i is either an atomic constraint (a *positive literal*) or a negative atomic constraint (a *negative literal*). For technical reasons, we will consider that the clause including only $\exists\emptyset$ (i.e. the true clause) is included in any specification. We will say that a clause is *strictly negative* if it only includes negative basic constraints.

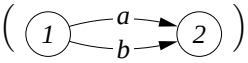
It may be noticed that dealing with arbitrary clauses is equivalent to deal with arbitrary boolean fomulas over the atomic constraints since these formulas can always be transformed into clausal form.

In the case of basic constraints the above definition specializes as expected:

Fact 1 ***(Satisfaction of basic constraints)*** *$G \models \exists C$ if there is a monomorphism $f : C \to G$.*

Remark 1 *Atomic constraints can be generalized by allowing its definition in terms of arbitrary morphisms. That is, we could have defined atomic graph constraints $\forall(c : X \to C)$ where c is an arbitrary morphism. However, with our notion of satisfaction, this generalization does not add any additional power to our logic, since it can be proved [HaP05] that if c is not a monomorphism then the constraint $\forall(c : X \to C)$ is logically equivalent to the constraint $\neg\exists X$. For instance, the two constraints below are equivalent. In particular, both constraints specify that there can not be two different edges between any two nodes.*

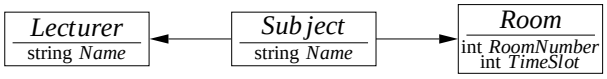
(1) **if** 

(2) 

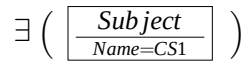
Analogously, we could have also generalized our notion of satisfaction by allowing h and f to be also arbitrary morphisms and not just monomorphisms. This generalized form of satisfaction has been studied in [HaP06], where it is called $\mathcal{A}$-satisfaction in contrast with the notion of satisfaction that we use, which is called $\mathcal{M}$-satisfaction in that paper. In particular, in [HaP06], it is shown how to transform nested constraints such that $\mathcal{A}$-satisfiability for a certain constraint is equivalent to $\mathcal{M}$-satisfiability for the transformed constraint (and vice versa). Anyhow, we believe that $\mathcal{M}$-satisfaction is more interesting than $\mathcal{A}$-satisfaction for specification purposes.

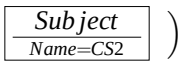
Remark 2 *The above notions can be defined not only for the category of graphs but for any weak adhesive HLR-category [LaS04, EEPT06] as can be seen in [EEHP04, EEPT06]. In particular, in that case, it is assumed that the morphisms involved in the notions of atomic constraints and satisfaction are not arbitrary monomorphisms but belong to a given class M of monomorphisms. In this context, the notions of constraints and satisfaction apply to many other kinds of graphical categories, including typed graphs and attributed typed graphs, as the ones considered in our running example.*

Example 1 *Let us suppose that we want to model an information system describing the lecturing organization of a department. Then the type graph of (part of) our system could be the following one:*

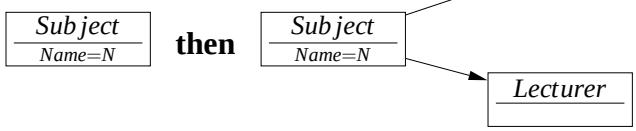


This means that in our system we have three types of nodes: Rooms including two attributes, the room number and a time slot, and Subjects and Lecturers, having their name as an attribute. We also have two types of edges. In particular, an edge from a Subject S to a Lecturer L means, obviously, that L is the lecturer for S. An edge from a Subject S to a Room means that the lecturing for S takes place on that room for the given time slot. Now for this system we could include the following constraints, where the type of each node is denoted by the word at the top of the square:

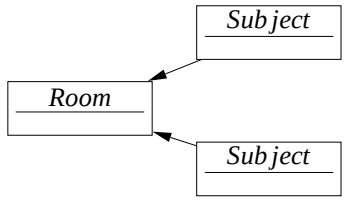
(1) 

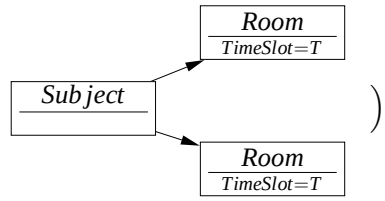
(2) 

meaning that the given system must include the compulsory subjects CS1 and CS2. Moreover we may have a constraint saying that every subject included in the system must have some lecturer assignment and some room assignment:

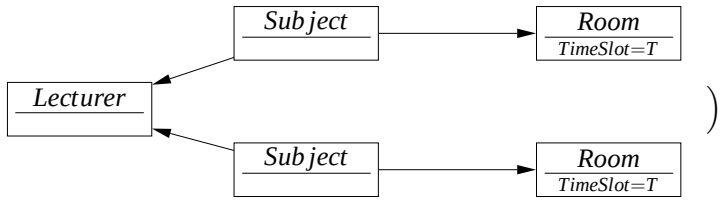
(3) **if** 

Then, we may also have constraints expressing some negative conditions. For instance, that a room is not assigned at the same time to two subjects or that two different rooms are assigned at the same time to the same subject:

(4) 

(5) 

or, similarly, that a lecturer does not have to lecture on two different subjects in two different rooms at the same time slot:

(6) 

Finally, perhaps we may want to specify that not every lecturer has a teaching assignment, so that every semester there may be someone on sabbatical:

(7) 

It may be noticed that the system that we are describing with these graphical constraints may not be an information system, but the set of web pages of a department, where an arrow from a node of type t_1 to a node of type t_2 may mean that there is a link between two web pages (for instance from the web page of a subject to the web pages of a lecturer), or it may mean that the information of type t_2 is a subfield of the information of type t_1 (for instance the room assignment may be a field of the subject's web pages). In this case, we could have displayed our constraints not in terms of graphs, but as HTML or XML expressions.

2.3. Refutation procedures for checking satisfiability

As it is often done in the area of automatic reasoning, the refutation procedures that we present in this paper are defined by means of some inference rules. More precisely, as usual, each rule tells us that if certain premises are satisfied then a given consequence will also hold. In this context, a refutation procedure can be seen as a (possibly nonterminating) nondeterministic computation where the current state is given by the set of formulas that have been inferred until the given moment, and where a computation step means adding to the given state the result of applying an inference rule to that state.

More precisely, in our case, we assume that in general the inference rules have the form:

$$\frac{\Gamma_1 \quad \Gamma_2}{\Gamma_3}$$

where Γ_1, Γ_2 and Γ_3 are clauses, and where clauses are seen as sets of literals. In particular, this means that if we write that a clause has the form $\Gamma \vee L$, this does not necessarily imply that L is the rightmost literal of the given clause. Similarly, we consider that the clause $\Gamma \vee L$ is the same as the clause $\Gamma \vee L \vee L$.

Then, a *refutation procedure* for a set of constraints $\mathcal{C}$ is a sequence of inferences:

$$\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_i \Rightarrow \ldots$$

where the initial state is the original specification (i.e., $\mathcal{C}_0 = \mathcal{C}$) and where we write $\mathcal{C}_i \Rightarrow \mathcal{C}_{i+1}$ if there is an inference rule like the one above such that $\Gamma_1, \Gamma_2 \in \mathcal{C}_i$, and $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\Gamma_3\}$. Moreover, we will assume that $\mathcal{C}_i \subset \mathcal{C}_{i+1}$, i.e. $\Gamma_3 \notin \mathcal{C}_i$, to avoid useless inferences.

In this framework, proving the unsatisfiability of a set of constraints means inferring the $false$ clause (which is represented by the empty clause, i.e. the empty disjunction, denoted $\Box$), provided that the procedure is sound and complete. Since the procedures are nondeterministic, there is the possibility that we never apply some key inference. To avoid this problem we will always assume that our procedures are *fair*, which means that, if at any moment i, there is a possible inference $\mathcal{C}_i \Rightarrow \mathcal{C}_i \cup \{\Gamma\}$, for some clause Γ, then at some moment j we have that $\Gamma \in \mathcal{C}_j$. This means that inferences are not postponed forever, i.e. every inference will eventually be performed. If we care about completeness, fairness must always be taken into account when implementing deduction. For instance, implementations based on depth-first search with backtracking run the risk of not being fair: if the deduction process gets into an infinite branch of the tree representing the deduction process, then it may fail to apply some alternative inferences. This is the well-known problem of the incompleteness of Prolog's implementation of SLD resolution [Llo87].

Then, a refutation procedure for $\mathcal{C}$ is *sound* if whenever the procedure infers the empty clause we have that $\mathcal{C}$ is unsatisfiable. And a procedure is *complete* if, whenever $\mathcal{C}$ is unsatisfiable, we have that the procedure infers $\Box$.

It may be noted that if a refutation procedure is sound and complete then we may know in a finite amount of time if a given set of constraints is unsatisfiable. However, it may be impossible to know in a finite amount of time if the set of constraints is satisfiable. For this reason, sometimes the above definition of completeness is called refutational completeness, using the term completeness when both satisfiability and unsatisfiability are decidable.

As usual, for proving soundness of a refutation procedure it is enough to prove the soundness of the inference rules. This means that for every rule as the one above and every graph G, if $G \models \Gamma_1$ and $G \models \Gamma_2$ then $G \models \Gamma_3$.

3. Basic Constraints and Positive Atomic Constraints

In this section we present an inference system consisting of the three rules (R1), (R2) and (R3) below that provides sound and complete refutation procedures for checking satisfiability when clauses consist only of positive and negative basic constraints and positive atomic constraints. This means that the given specifications are assumed to consist of clauses including literals of the form $\exists C_1$, $\neg\exists C_1$, or $\forall(c : X \to C_2)$.

Our refutation procedures may not terminate, which means that the procedures are just refutationally complete. However, as shown in [OEP08], if we restrict our logic to basic constraints then refutation procedures would terminate. Moreover, our procedures check satisfiability with respect to the class of finite and infinite graphs. In fact, in the following section, we show an example of a specification whose only models are infinite graphs. As a consequence, we guess that satisfiability for this class of constraints is already undecidable (but semi-decidable).

$$\frac{\exists C_1 \vee \Gamma_1 \quad \neg\exists C_2 \vee \Gamma_2}{\Gamma_1 \vee \Gamma_2} \quad \textbf{(R1)}$$

if there exists a monomorphism $m : C_2 \to C_1$

$$\frac{\exists C_1 \vee \Gamma_1 \quad \exists C_2 \vee \Gamma_2}{(\bigvee_{G \in \mathcal{G}} \exists G) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R2)}$$

where $\mathcal{G} = \{G \mid \langle f_1 : C_1 \to G \leftarrow C_2 : f_2 \rangle \in (C_1 \otimes C_2)\}$ and $(\bigvee_{G \in \mathcal{G}} \exists G)$ denotes the (finite) disjunction $\exists G_1 \vee \cdots \vee \exists G_n$, if $\mathcal{G} = \{G_1, \ldots, G_n\}$.

$$\frac{\exists C_1 \vee \Gamma_1 \quad \forall(c : X \to C_2) \vee \Gamma_2}{(\bigvee_{G \in \mathcal{G}} \exists G) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R3)}$$

if there is a monomorphism $m : X \to C_1$ and $\mathcal{G} = \{G \mid \langle f_1 : C_1 \to G \leftarrow C_2 : f_2 \rangle \in (C_1 \otimes C_2)$ such that $f_1 \circ m = f_2 \circ c\}$.

The first rule is, in some sense, similar to resolution and is the rule that may allow us to infer the empty clause. The reason is that it is the only rule that eliminates literals from clauses. The second one can be seen as a rule that, given two constraints, builds a new constraint that subsumes them. More precisely, the graphs involved in the new literals in the clause, i.e. the graphs $G \in \mathcal{G}$ satisfy both constraints $\exists C_1$ and $\exists C_2$. This means that if we apply this rule

repeatedly, using all the positive constraints in the original set $\mathcal{C}$, we would build graphs that satisfy all the positive basic constraints in $\mathcal{C}$. The third rule is similar to rule (R2) in the sense that given a positive basic constraint and a positive atomic constraint it builds a disjunction of literals representing graphs that try to satisfy both constraints. However, in this case the satisfaction of the constraint $\forall(c : X \rightarrow C_2)$ is not necessarily ensured for all $G \in \mathcal{G}$. In particular, the idea of the rule is that if we know that X is included in C_1 then we build all the possible extensions of C_1 which also include C_2 (each G would be one of such extensions). But we cannot be sure that G satisfies $\forall(c : X \rightarrow C_2)$, because G may include an instance of X which was not included in C_1. For instance, suppose that we have the following constraints:

(1) $\exists\left(\ \bigcirc\ \right)$ **(2)** **if** (a) **then** (a) → (b)

where the first one specifies that the given graph must include a node and where the second one specifies that every node must have an outgoing edge. Then applying rule (R3) to these constraints would yield a clause including the literal:

$\exists\left(\ (a) \rightarrow (b)\ \right)$

Now, in this graph, the node *a* has an outgoing edge, but the node *b* does not have it, so the graph still does not satisfy the second constraint. If we would apply again the third rule, then we would infer a clause including a graph with three nodes and two edges, and so on. This is the reason why, in this case, a refutation procedure may not terminate. Moreover, as we will also see, if the procedure does not refute the given set of constraints then the completeness proof ensures that there will be a model that satisfies this set of constraints, but this model may be an infinite graph built by an infinite colimit. One may wonder whether there will also exist a finite model of that specification. In the case of this example such a finite graph exists. Actually, the resulting clause after applying for the second time the third rule to the graph above, would also include the graph below that satisfies both constraints.



However, in general, we do not know if an arbitrary set of basic constraints and positive atomic constraints which is satisfiable by an infinite graph, is also satisfied by some finite graph. Nevertheless, in the general case (when dealing with positive and negative atomic constraints) there are sets of constraints whose only models are infinite graphs, as we will see in the following section. For this reason we conjecture that in this case the answer to this question will also be negative.

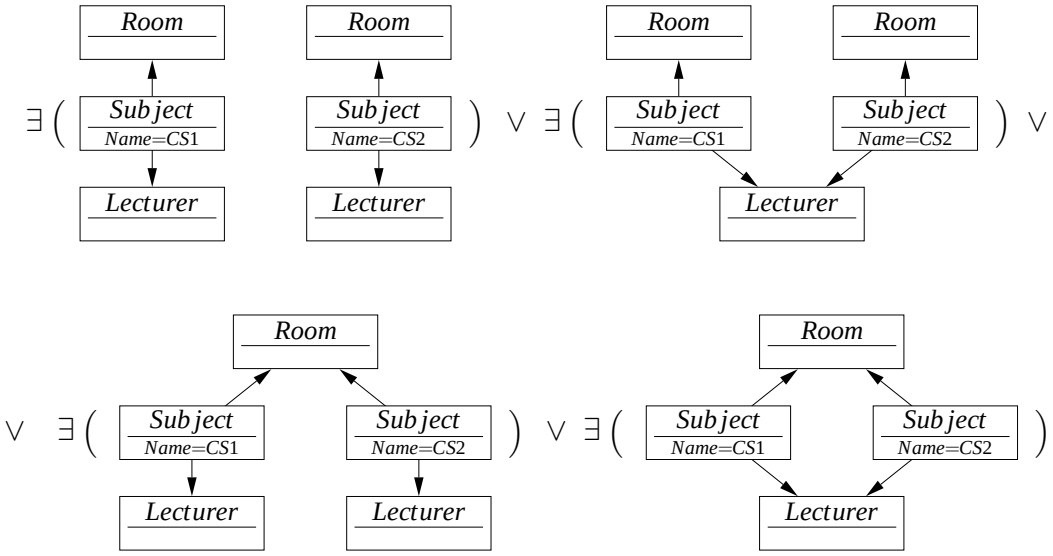
Example 2 *If we consider the basic constraints and the positive atomic constraints that are included in the Example 1 (i.e. the constraints (1), (2), (3), (4), (5), and (6)) then it would first be possible to infer the constraint below using the rule (R2) on constraints (1) and (2):*[1]

(8) $\exists\left(\ \boxed{\begin{array}{c}Subject\\ \hline Name{=}CS1\end{array}}\qquad \boxed{\begin{array}{c}Subject\\ \hline Name{=}CS2\end{array}}\ \right)$

This new constraint obviously means that the graph representing the system must include at least two Subject nodes (with attributes CS1 and CS2). Then, if we apply the third rule on constraints (8) and (3), and, again, on the resulting clause and on constraint (3) then we would infer the following clause:

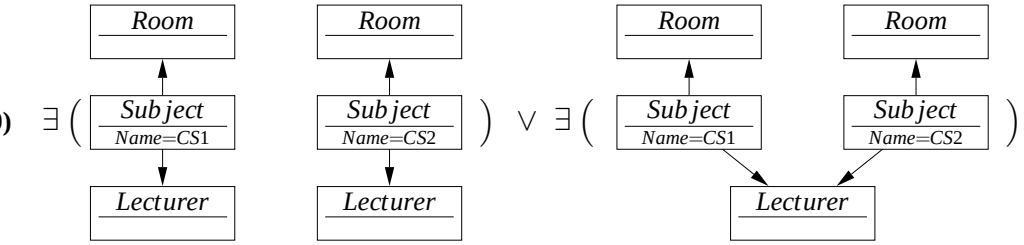
[1] Actually, if the graphs in the example would be considered attributed graphs as presented in [EEPT06], then the clause inferred would include an additional literal. In particular this literal would be a graph consisting of a single node of type *Subject* with two *Name* attributes. However, from now on, in our examples we will assume that it is not possible that a node has twice the same attribute. This could be done, for instance, assuming that our specifications implicitly include a graph constraint stating that this situation is not allowed.

(9)



This clause states that the graph should include two subjects (CS1 and CS2) and these subjects may be assigned to two different rooms and to either two different lecturers, or to the same lecturer, or they may be assigned to the same room, and to either different lecturers, or the same lecturer. Obviously, the last two constraints in this clause violate constraint (4), which means that we can eliminate them using twice the rule (R1), yielding the following clause:

(10)



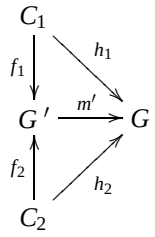
At this point, we could stop the inference process since the two graphs in (10) are already (minimal) models of the given set of constraints, which means that it is satisfiable. Actually, the inferences that we can apply to the current set of clauses are quite useless and we could have avoided them by defining more restrictive side conditions in the inference rules. For instance, in (R2) we could have asked, in addition, that there should not exist any monomorphism from C_1 to C_2 or vice versa since it could be proved that, if such monomorphism exists, the deduction rule is useless. However, we have preferred to present this (inefficient) version of the deduction rules to simplify as much as possible the completeness proof.

It is easy to prove that these three rules are sound:

Lemma 1 ***(Soundness of the inference rules)*** *Rules (R1), (R2), and (R3) are sound.*

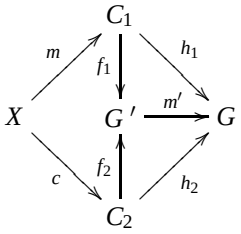
Proof. (R1) Let G be a graph and suppose that $G \models \exists C_1 \vee \Gamma_1$, $G \models \neg\exists C_2 \vee \Gamma_2$, and there exists a monomorphism $m : C_2 \to C_1$. We know that it cannot happen that $G \models \exists C_1$ and $G \models \neg\exists C_2$, since if $G \models \exists C_1$ then there exists a monomorphism $h : C_1 \to G$ and this implies that $h \circ m : C_2 \to G$ is a monomorphism, meaning that $G \models \exists C_2$. Therefore, $G \models \Gamma_1 \vee \Gamma_2$.

(R2) Suppose that $G \models \exists C_1 \vee \Gamma_1$ and $G \models \exists C_2 \vee \Gamma_2$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Suppose that $G \models \exists C_1$ and $G \models \exists C_2$. This means that there are two monomorphisms $h_1 : C_1 \to G$ and $h_2 : C_2 \to G$ and this implies by Prop. 1 that there is a factorization:



where $f_1 : C_1 \to G'$ and $f_2 : C_2 \to G'$ are jointly surjective, which means that G' is in $\mathcal{G}$, and m' is injective. This implies that $G \models (\bigvee_{G' \in \mathcal{G}} \exists G')$

(R3) Suppose that $G \models \exists C_1 \vee \Gamma_1$, $G \models \forall(c : X \to C_2) \vee \Gamma_2$, and there is a monomorphism $m : X \to C_1$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Suppose that $G \models \exists C_1$ and $G \models \forall(c : X \to C_2)$, this means, on one hand, that there is a monomorphism $h_1 : C_1 \to G$. On the other hand, this also means that there is a monomorphism $h_2 : C_2 \to G$ such that $h_1 \circ m = h_2 \circ c$, since $G \models \forall(c : X \to C_2)$. As a consequence, by Prop. 2 there is a factorization:



where $f_1 : C_1 \to G'$ and $f_2 : C_2 \to G'$ are jointly surjective, which means that G' is in $\mathcal{G}$, and m' is injective. This implies that $G \models (\bigvee_{G' \in \mathcal{G}} \exists G')$ □

Proving completeness is more involved. The underlying idea of the completeness proof is to consider a precedence relation between the basic literals (or the associated graphs) occurring in clauses. Then, we will show that the colimit of one of these sequences is the model of the given specification. More precisely, we will see that the sequences considered represent a construction of possible models using the inference rules (R2) and (R3). But before proving the completeness of our system, let us first present some auxiliary definitions and results. We start by defining a key construction for proving completeness, related to inference rules (R2) and (R3). Given a basic constraint, $\exists G_1$, and a positive literal, L, $I(\exists G_1, L)$ is the set of all literals $\exists G$ (or rather of morphisms $h : G_1 \to G$) that can be inferred from $\exists G_1$ and L using the rules (R2) or (R3). In particular, in the case where $L = \exists G_2$, this means, essentially, $G_1 \otimes G_2$. In the case where the second literal is $\forall(c : X \to G_2)$ we iterate the construction for each of the monomorphisms from X to G_1.

We may notice that, in the proofs below, we do not make explicit use of the fairness requirement for the given refutation procedures. However this requirement is implicitly used in a number of proofs. More precisely, given a refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$, whenever we are assuming that the result of a certain inference is in $\bigcup_{k \geq 1} \mathcal{C}_k$ (assuming, obviously, that the premises are also in $\bigcup_{k \geq 1} \mathcal{C}_k$) we are implicitly assuming that the procedure is fair.

Definition 6 *Let $\exists G_1$ be a basic literal and L a positive literal. We define the set of monomorphisms $I(\exists G_1, L)$ by cases:*

- *If L is a basic literal, $L = \exists G_2$, then $I(\exists G_1, L) = \{ f_1 \mid \langle f_1 : G_1 \to G \leftarrow G_2 : f_2 \rangle \in (G_1 \otimes G_2) \}$.*
- *If L is a non-basic literal, $L = \forall(c : X \to C)$, and H is the set of all monomorphisms from X to G_1 then $I(\exists G_1, L) = I^*(\exists G_1, H)$, where $I^*(\exists G_1, H)$ is defined inductively:*
  - *If $H = \emptyset$ then $I^*(\exists G_1, H) = \{id_{G_1}\}$, where id_{G_1} denotes the identity, $id_{G_1} : G_1 \to G_1$*

– *If* $H = \{f : X \to G_1\} \cup H'$ *then* $I^*(\exists G_1, H) = \{h' \circ h \mid h : G_1 \to G \in I^*(\exists G_1, H'), \langle h' : G \to G' \leftarrow C : f_2 \rangle \in (G \otimes C)$ *such that* $f_2 \circ c = h' \circ h \circ f\}$.

Notice that, by Prop. 3, the above definition is independent of the order in which we consider the monomorphisms in H.

The definition of I is extended to clauses and sets of clauses. $I(\exists G_1, \Gamma)$ is the set of all literals $\exists G$ (morphisms $h : G_1 \to G$) that can be inferred from $\exists G_1$ and the positive literals in Γ using the rules (R2) or (R3). Then, $I(\exists G_1, \mathcal{C})$ is the set of all literals $\exists G$ (morphisms $h : G_1 \to G$) that can be inferred from $\exists G_1$ after applying one inference with each of the clauses (one after the other) in $\mathcal{C}$. However, if a clause Γ in $\mathcal{C}$ includes a negative literal, $\neg\exists G$ and we have that $G_1 \models \exists G$ then no inference would be applied when computing I, since G_1 would already satisfy Γ. The same happens if Γ includes a literal $\forall(c : X \to G_2)$ and there is no monomorphism $h : X \to G_1$. Notice that this implies, as we can see in the definition below, that if Γ is a strictly negative clause then $I(\exists G_1, \Gamma)$ is the empty set. Also, if $\Gamma \in \mathcal{C}$ is strictly negative and for every literal $\neg\exists C$ in Γ we have that there is a monomorphism from C into G_1 then $I(\exists G_1, \mathcal{C})$ is again the empty set. Otherwise, if G_1 satisfies all the strictly negative clauses in $\mathcal{C}$ then $I(\exists G_1, \mathcal{C})$ is not empty.

Definition 7 *Let* $\exists G_1$ *be a basic literal and* Γ *be a clause. We define the set of monomorphisms* $I(\exists G_1, \Gamma)$*:*

$$I(\exists G_1, \Gamma) = \bigcup_{\exists C \in \Gamma} I(\exists G_1, \exists C) \;\cup \bigcup_{\forall(c : X \to C) \in \Gamma} I(\exists G_1, \forall(c : X \to C))$$

If $\exists G_1$ *is a basic literal and* $\mathcal{C}$ *is a set of clauses. We define the set of monomorphisms* $I(\exists G_1, \mathcal{C})$ *inductively:*

- *If* $\mathcal{C}$ *is the empty set, then* $I(\exists G_1, \mathcal{C}) = \{id_{G_1}\}$.
- *If* $\mathcal{C} = \{\Gamma\} \cup \mathcal{C}'$*, and* Γ *includes a negative literal,* $\neg\exists G$ *such that* $G_1 \models \neg\exists G$*, or* Γ *includes a positive atomic literal,* $\forall(c : X \to G_2)$ *such that there is no monomorphism* $h : X \to G_1$ *then* $I(\exists G_1, \mathcal{C}) = I(\exists G_1, \mathcal{C}')$.
- *Otherwise,* $I(\exists G_1, \{\Gamma\} \cup \mathcal{C}') = \{g \circ h \mid g \in I(\exists G, \Gamma), (h : G_1 \to G) \in I(\exists G_1, \mathcal{C}')\}$.

Given a literal G_1 and a set of clauses $\mathcal{C}$, the relation between the set $I(\exists G_1, \mathcal{C})$ and our inference rules is made explicit by the following propositions. In particular, the aim of these two propositions is to show that if a literal $\exists G_2$ is in $I(\exists G_1, \mathcal{C})$ then, in every clause Γ in $\mathcal{C}$ there should be a literal L such that $\exists G_2$ can be seen as one of the results of an inference of $\exists G_1$ and L. First we consider the case where $\mathcal{C}$ consists of a single clause.

Proposition 5 *Let* Γ *be a clause consisting of basic constraints and positive atomic constraints and let* $h : G_1 \to G_2$ *be a monomorphism such that* $h \in I(\exists G_1, \Gamma)$*, then there is a literal* L *in* Γ *such that:*

- *if* $L = \neg\exists C$*, then there is no monomorphism* $m : C \to G_1$.
- *if* $L = \exists C$*, then there is a monomorphism* $m : C \to G_2$.
- *If* $L = \forall(c : X \to C)$ *then for every monomorphism* $f : X \to G_1$ *there is a monomorphism* $g : C \to G_2$ *with* $h \circ f = g \circ c$.

Proof. By definition, we know that $I(\exists G_1, \Gamma) = \bigcup_{\exists C \in \Gamma} I(\exists G_1, \exists C) \cup \bigcup_{\forall(c : X \to C) \in \Gamma} I(\exists G_1, \forall(c : X \to C))$. We consider several cases:

- If Γ does not include any positive literal (i.e. Γ is the empty clause or Γ includes only negative literals), then the proposition trivially holds, since by definition $I(\exists G_1, \Gamma)$ is empty.
- If Γ includes a negative literal, $\neg\exists C$ such that $G_1 \models \neg\exists C$ then the proposition trivially holds, since it is enough to take $L = \neg\exists C$.
- If Γ includes an atomic literal $\forall(c : X \to C)$ and there is no monomorphism h from X to G_1, then the proposition trivially holds, since it is enough to take $L = \forall(c : X \to C)$.
- If $(h : G_1 \to G_2) \in \bigcup_{\exists C \in \Gamma} I(\exists G_1, \exists C)$, then $h \in I(\exists G_1, \exists C)$ for some literal $\exists C$ in Γ. Then, by definition of $I(\exists G_1, \exists C)$, $(h : G_1 \to G_2) \in \{f_1 \mid \langle f_1 : G_1 \to G_2 \leftarrow C : f_2 \rangle \in (G_1 \otimes C)\}$, which means that there is a monomorphism $f_2 : C \to G_2$.
- If $(h : G_1 \to G_2) \in \bigcup_{\forall(c : X \to C) \in \Gamma} I(\exists G_1, \forall(c : X \to C))$, then $h \in I(\exists G_1, L)$ for some literal $L = \forall(c : X \to C)$ in Γ. Let $H = \{f_1, \ldots, f_n\}$ be the set of all monomorphisms from X to G_1. Then, by definition, we know that the monomorphisms in $I(\exists G_1, \forall(c : X \to C))$ are defined as compositions $h_n \circ \cdots \circ h_1$, where $h_1 : G_1 \to C_1$ and, for each i, $h_{i+1} : C_i \to C_{i+1}$, $h_i \circ \cdots \circ h_1 : G_1 \to C_i \in I^*(\exists G_1, \{f_1, \ldots, f_i\}$ and there is a monomorphism $(g_i : C \to C_i)$, such

that $g_i \circ c = h_i \circ \cdots \circ h_1 \circ f_i$. Therefore, given a monomorphism $f_j : X \to G_1$, we have that $g_j \circ c = h_j \circ \cdots \circ h_1 \circ f_j$ and this means that if we define $g : C \to G_2$ as $g = h_n \circ \cdots \circ h_{j+1} \circ g_j$, then $g \circ c = h_n \circ \cdots \circ h_{j+1} \circ g_j \circ c = h_n \circ \cdots \circ h_{j+1} \circ h_j \circ \cdots \circ h_1 \circ f_j = h \circ f_j$.

□

Now, we extend the previous result to an arbitrary set of clauses $\mathcal{C}$.

Proposition 6 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints and let $h : G_1 \to G_2$ be a monomorphism such that $h \in I(\exists G_1, \mathcal{C})$, then for every clause Γ in $\mathcal{C}$ there is a literal L in Γ such that:*

- *if $L = \neg\exists C$, then there is no monomorphism $m : C \to G_1$.*
- *if $L = \exists C$, then there is a monomorphism $m : C \to G_2$.*
- *If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_1$ there is a monomorphism $f : C \to G_2$ with $h \circ m = f \circ c$.*

Proof. We prove the proposition by induction on $\mathcal{C}$, following the definition of $I(\exists G_1, \mathcal{C})$:

- If $\mathcal{C}$ is the empty set, then the proposition trivially holds.
- Otherwise, if $\mathcal{C} = \{\Gamma\} \cup \mathcal{C}'$, by induction, we know that if $h' : G_1 \to G_2' \in I(\exists G_1, \mathcal{C}')$ every Γ' in $\mathcal{C}'$ satisfies the proposition with respect to h'. Therefore, if $h = g \circ h' \in I(\exists G_1, \mathcal{C})$, with $g : G_2' \to G_2 \in I(\exists G_2', \Gamma)$, on one hand we have to prove that every Γ' in $\mathcal{C}'$ satisfies the proposition with respect to $g \circ h'$ and, on the other, that Γ also satisfies the proposition with respect to $g \circ h'$.
  Given a clause Γ' in $\mathcal{C}'$, by induction, we know that there is a a literal L in Γ such that one of the following cases holds:
  - if $L = \neg\exists C$, the case is trivial.
  - if $L = \exists C$, then there is a monomorphism $m : C \to G_2$. But this means that $g \circ m : C \to G_2'$
  - If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_1$ there is a monomorphism $f : C \to G_2'$ with $h' \circ m = f \circ c$. But this means that there is a monomorphism $g \circ f : C \to G_2$. Moreover, $g \circ f \circ c = g \circ h' \circ m = h \circ m$

  Let us now consider the clause Γ. We have the following cases:
  - If Γ includes a negative literal, $\neg\exists G$ such that $G_1 \models \neg\exists G$ then the proposition trivially holds, since it is enough to take $L = \neg\exists G$.
  - Γ includes a positive atomic literal, $\forall(c : X \to G_2)$ such that there is no monomorphism $h : X \to G_1$ then again the proposition trivially holds, since it is enough to take $L = \forall(c : X \to G_2)$.
  - Otherwise, by Prop. 5, we know that there is a literal L in Γ such that
    - if $L = \exists C$, then there is a monomorphism $m : C \to G_2$.
    - If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_2'$ there is a monomorphism $f : C \to G_2$ with $g \circ m = f \circ c$. Suppose, in this case, that we have a monomorphism $m' : X \to G_1$, then this means that we have a monomorphism $h' \circ m' : X \to G_2'$, therefore there should exist a monomorphism $f : C \to G_2$ with $f \circ c = g \circ h' \circ m' = h \circ m'$.

□

A direct consequence of the proposition above is that if the identity morphism is in $I(\exists G_1, \mathcal{C})$ then G_1 is a model of $\mathcal{C}$.

Proposition 7 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints, if $id_G \in I(\exists G, \mathcal{C})$, then $G \models \mathcal{C}$.*

Proof. If $id_G \in I(\exists G, \mathcal{C})$ then, according to Prop. 6, for every Γ in $\mathcal{C}$ there is a literal L in Γ such that:

- if $L = \neg\exists C$, then there is no monomorphism $m : C \to G$. But this means that $G \models L$ and, hence, $G \models \Gamma$.
- if $L = \exists C$, then there is a monomorphism $m : C \to G$. Therefore $G \models L$ and, hence, $G \models \Gamma$.
- If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G$ there is a monomorphism $f : C \to G$ with $id_G \circ m = m = f \circ c$. Again, this means that $G \models L$ and, hence, $G \models \Gamma$.

Therefore G satisfies all the clauses in $\mathcal{C}$. $\square$

The aim of the next two propositions is to show that if we have a literal $\exists G_1$ in a clause Γ, then we can infer the clause resulting from replacing that literal by a disjunction consisting of all the graphs in $I(\exists G_1, \mathcal{C})$.

Proposition 8 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints, let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3) and let $\exists G_1 \vee \Gamma_1$ and Γ_2 be two non-empty clauses in $\bigcup_{k\geq 1} \mathcal{C}_k$ such that for every negative literal $\neg\exists G_2 \in \Gamma_2$ we have that $G_1 \not\models \neg\exists G_2$ and for every atomic literal $\forall(c : X \to G_2) \in \Gamma_2$ there is a monomorphism $h : X \to G_1$. Then $(\bigvee_{G\in\mathcal{G}} \exists G \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G} = \{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\}$.*

Proof. We prove something slightly more general: that for all clauses Γ_1 and Γ_2, if $\exists G_1 \vee \Gamma_1$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$ and for every literal $\neg\exists G_2 \in \Gamma_2$ we have that $G_1 \not\models \neg\exists G_2$ and for every literal $\forall(c : X \to G_2) \in \Gamma_2$ there is a monomorphism $h : X \to G_1$, then for every clause Γ_3 such that $\Gamma_2 \vee \Gamma_3$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$ we have that either Γ_2 is empty or $(\bigvee_{G\in\mathcal{G}} \exists G \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G} = \{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\}$. Note that to prove the proposition it is enough to consider that Γ_3 is the empty clause. We prove this by induction on Γ_2.

If Γ_2 is the empty clause, then the proof is trivial. Otherwise, let us suppose that $\Gamma_2 = L \vee \Gamma_2'$, for a given literal L. By induction, we may assume that for any Γ_3 if $(\Gamma_2' \vee L \vee \Gamma_3) \in \bigcup_{k\geq 1} \mathcal{C}_k$ then either (a) Γ_2' is empty or (b) $(\bigvee_{G\in\mathcal{G}'} \exists G \vee L \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}' = \{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2')\}$. Let us define Γ_3' to be equal to Γ_3 in case (a), and equal to $(\bigvee_{G\in\mathcal{G}'} \exists G \vee \Gamma_3 \vee \Gamma_1)$ in case (b). We have three cases:

- If $L = \neg\exists G_2$ then we know that $\neg\exists G_2 \vee \Gamma_3' \in \bigcup_{k\geq 1} \mathcal{C}_k$. By assumption, we know that $G_1 \not\models \neg\exists G_2$, which means that there is a morphism from G_2 to G_1. Thus, we can apply rule (R1) to $\exists G_1 \vee \Gamma_1$ and $\neg\exists G_2 \vee \Gamma_3'$ yielding the clause $\Gamma_3' \vee \Gamma_1 \in \bigcup_{k\geq 1} \mathcal{C}_k$. Therefore, in case (a) we know that $\Gamma_3 \vee \Gamma_1 \in \bigcup_{k\geq 1} \mathcal{C}_k$, and this completes the proof, since in this case, by definition, $\{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\} = \emptyset$. On the other hand, in case (b) we know that $(\bigvee_{G\in\mathcal{G}'} \exists G \vee \Gamma_3 \vee \Gamma_1) \vee \Gamma_1 \equiv (\bigvee_{G\in\mathcal{G}'} \exists G \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$ and this completes also the proof, since in this case, by definition, $\{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\} = \{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2')\}$.
- If $L = \exists G_2$ then we know that $\exists G_2 \vee \Gamma_3' \in \bigcup_{k\geq 1} \mathcal{C}_k$. Thus, we can apply rule (R2) to $\exists G_1 \vee \Gamma_1$ and $\exists G_2 \vee \Gamma_3'$ yielding the clause $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \Gamma_3') \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}'' = \{G \mid \langle f_1 : G_1 \to G \leftarrow G_2 : f_2\rangle \in (G_1 \otimes G_2)\}$. Therefore, in case (a) we know that $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \Gamma_3) \in \bigcup_{k\geq 1} \mathcal{C}_k$, and this completes the proof, since in this case, by definition, $\{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\} = \{G \mid \langle f_1 : G_1 \to G \leftarrow G_2 : f_2\rangle \in (G_1 \otimes G_2)\}$. On the other hand, in case (b) we know that $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \bigvee_{G\in\mathcal{G}'} \exists G \vee \Gamma_3 \vee \Gamma_1) \equiv (\bigvee_{G\in\mathcal{G}''} \exists G \vee \bigvee_{G\in\mathcal{G}'} \exists G \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$ and this completes the proof, since in this case, by definition, $\{G \mid h : G_1 \to G \in I(\exists G_1, \Gamma_2)\} = \mathcal{G}'' \cup \mathcal{G}'$.
- If $L = \forall(c : X \to G_2)$ then we know that $\forall(c : X \to G_2) \vee \Gamma_3' \in \bigcup_{k\geq 1} \mathcal{C}_k$. Let H be the set of all monomorphisms from X to G_1, which by assumption is not empty. By definition, $I(\exists G_1, \forall(c : X \to G_2)) = I^*(\exists G_1, H)$. So we will prove by induction that, for any non-empty H, $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \Gamma_3')$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}'' = \{G \mid f : G_1 \to G \in I^*(\exists G_1, H)\}$.
  - If $H = \{g : X \to G_1\}$ then applying rule (R3) to $\exists G_1 \vee \Gamma_1$ and $(\forall(c : X \to G_2) \vee \Gamma_3')$ we infer $(\bigvee_{G\in\mathcal{G}'''} \exists G \vee \Gamma_1 \vee \Gamma_3')$, where $\mathcal{G}''' = \{G \mid \langle f_1 : G_1 \to G \leftarrow G_2 : f_2\rangle \in (G_1 \otimes G_2)$ such that $f_1 \circ g = f_2 \circ c\}$. But, in this case, $\mathcal{G}'' = \mathcal{G}'''$. Hence, $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \Gamma_3')$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$.
  - If $H = \{g : X \to G_1\} \cup H'$ then, by induction, we may assume that $(\bigvee_{G\in\mathcal{G}'''} \exists G \vee \Gamma_1 \vee \Gamma_3')$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}''' = \{G \mid f : G_1 \to G \in I^*(\exists G_1, H')\}$. Let us assume that $I^*(\exists G_1, H') = \{f_1 : G_1 \to C_1, \ldots, f_n : G_1 \to C_n\}$, i.e. $\mathcal{G}''' = \{C_1, \ldots, C_n\}$ and $(\bigvee_{G\in\mathcal{G}'''} \exists G \vee \Gamma_1 \vee \Gamma_3') \equiv \exists C_1 \vee \cdots \vee \exists C_n \vee \Gamma_1 \vee \Gamma_3'$. We know that for every i we have a monomorphism $f_i \circ g : X \to C_i$. Therefore, we can apply rule (R3) to $\exists C_1 \vee \cdots \vee \exists C_n \vee \Gamma_1 \vee \Gamma_3'$ and to $(\forall(c : X \to G_2) \vee \Gamma_3')$ inferring the clause $((\bigvee_{G\in\mathcal{G}_1} \exists G) \vee \exists C_2 \vee \cdots \vee \exists C_n \vee \Gamma_1 \vee \Gamma_3' \vee \Gamma_3') \equiv ((\bigvee_{G\in\mathcal{G}_1} \exists G) \vee \exists C_2 \vee \cdots \vee \exists C_n \vee \Gamma_1 \vee \Gamma_3'$, where $\mathcal{G}_1 = \{G \mid \langle h_1 : C_1 \to G \leftarrow G_2 : h_2\rangle \in (C_1 \otimes G_2)$ such that $h_1 \circ f_1 \circ g = h_2 \circ c\}$. Now, if we apply again rule (R3) to the previous clause and to $(\forall(c : X \to G_2) \vee \Gamma_3')$ and we repeat this process n times, applying the rule to each of the literals C_i, we would finally infer the clause $((\bigvee_{G\in\mathcal{G}_1} \exists G) \vee \cdots \vee (\bigvee_{G\in\mathcal{G}_n} \exists G) \vee \Gamma_1 \vee \Gamma_3')$, where $\mathcal{G}_i = \{G \mid \langle h_1 : C_i \to G \leftarrow G_2 : h_2\rangle \in (C_1 \otimes G_2)$ such that $h_1 \circ f_i \circ g = h_2 \circ c\}$. This means that $((\bigvee_{G\in\mathcal{G}'''} \exists G) \vee \Gamma_1 \vee \Gamma_3') \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}''' = \{G \mid \langle h_1 : C_i \to G \leftarrow G_2 : h_2\rangle \in (C_i \otimes G_2)$ such that $(f_i : G_1 \to C_i) \in I^*(\exists G_1, H')$ and $h_1 \circ f_i \circ g = h_2 \circ c\}$. But, by definition, $\mathcal{G}''' = \{G \mid f : G_1 \to G \in I^*(\exists G_1, H)\}$. Therefore, we have also proved in this case that $(\bigvee_{G\in\mathcal{G}''} \exists G \vee \Gamma_1 \vee \Gamma_3')$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}'' = \{G \mid f : G_1 \to G \in I^*(\exists G_1, H)\}$.

Hence, in case (a) we know that $(\bigvee_{G\in\mathcal{G}''}\exists G\vee\Gamma_1\vee\Gamma_3)\in\bigcup_{k\geq 1}\mathcal{C}_k$, and this completes the proof, since in this case, by definition, $\{G\mid h:G_1\to G\in I((\forall(c:X\to G_2),\Gamma_2)\}=\{G\mid f:G_1\to G\in I^*(\exists G_1,H)\}$. On the other hand, in case (b) we know that $((\bigvee_{G\in\mathcal{G}''}\exists G)\vee\Gamma_1\vee(\bigvee_{G\in\mathcal{G}'}\exists G)\vee\Gamma_3\vee\Gamma_1)\equiv((\bigvee_{G\in\mathcal{G}''}\exists G(\vee(\bigvee_{G\in\mathcal{G}'}\exists G)\vee\Gamma_3\vee\Gamma_1)\in\bigcup_{k\geq 1}\mathcal{C}_k$ and this completes the proof, since in this case, by definition, $\{G\mid h:G_1\to G\in I((\forall(c:X\to G_2),\Gamma_2)\}=\mathcal{G}''\cup\mathcal{G}'$.

□

The above proposition can be extended as follows:

Proposition 9 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints, let $\mathcal{C}\Rightarrow\mathcal{C}_1\Rightarrow\cdots\Rightarrow\mathcal{C}_k\ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3) and let $\exists G_1\vee\Gamma_1$ be a clause in $\bigcup_{k\geq 1}\mathcal{C}_k$, then for any $\mathcal{C}'\subseteq\mathcal{C}$, $(\bigvee_{G\in\mathcal{G}}\exists G\vee\Gamma_1)\in\bigcup_{k\geq 1}\mathcal{C}_k$, where $\mathcal{G}=\{G\mid h:G_1\to G\in I(\exists G_1,\mathcal{C}')\}$.*

Proof. We proof the proposition by induction:

- If $\mathcal{C}'$ is empty then the case is trivial since $I(\exists G_1,\mathcal{C}')=\{id_{G_1}\}$ and, hence, $(\bigvee_{G\in\mathcal{G}}\exists G\vee\Gamma_1)=\exists G_1\vee\Gamma_1$.
- If $\mathcal{C}'=\{\Gamma\}\cup\mathcal{C}''$, and Γ includes a negative literal, $\neg\exists G$ such that $G_1\models\neg\exists G$ or Γ includes a positive atomic literal, $\forall(c:X\to G_2)$ such that there is no monomorphism $h:X\to G_1$, then the case is also trivial, since by definition $I(\exists G_1,\mathcal{C}')=I(\exists G_1,\mathcal{C}'')$ and, by induction, we may assume that $(\bigvee_{G\in\mathcal{G}}\exists G\vee\Gamma_1)\in\bigcup_{k\geq 1}\mathcal{C}_k$, where $\mathcal{G}=\{G\mid h:G_1\to G\in I(\exists G_1,\mathcal{C}'')\}$.
- If $\mathcal{C}'=\{\Gamma\}\cup\mathcal{C}''$, for every negative literal $\neg\exists G$ in Γ we have that $G_1\not\models\neg\exists G$ (i.e. there is a monomorphism $h_1:G\to G_1$) and for every atomic literal $\forall(c:X\to G_2)$ in Γ there is a monomorphism $h_2:X\to G_1$, then by definition we know that $I(\exists G_1,\{\Gamma\}\cup\mathcal{C}'')=\{g\circ h\mid g\in I(\exists G,\Gamma),(h:G_1\to G)\in I(\exists G_1,\mathcal{C}'')\}$. This means that we have to prove that $(\bigvee_{G\in\mathcal{G}}\exists G\vee\Gamma_1)\in\bigcup_{k\geq 1}\mathcal{C}_k$, where $\mathcal{G}=\{G\mid g\circ h:G_1\to G,g\in I(\exists G,\Gamma),(h:G_1\to G)\in I(\exists G_1,\mathcal{C}'')\}$. This is equivalent to prove that $(\bigvee_{G'\in\mathcal{G}'}\bigvee_{G\in\mathcal{G}'_G}\exists G)\vee\Gamma_1\in\bigcup_{k\geq 1}\mathcal{C}_k$, where $\mathcal{G}'=\{G'\mid h:G_1\to G'\in I(\exists G_1,\mathcal{C}'')\}$ and $\mathcal{G}_{G'}=\{G\mid g:G'\to G,g\in I(\exists G',\Gamma)\}$.
  By induction we know that $(\bigvee_{G'\in\mathcal{G}'}\exists G'\vee\Gamma_1)\in\bigcup_{k\geq 1}\mathcal{C}_k$. We also know that for every $G'\in\mathcal{G}'$ and for every negative literal $\neg\exists G$ in Γ we have that there is a monomorphism from G to G', since we know that there is a monomorphism from G to G_1 and also from G_1 to G'. And, in addition, we know that for every atomic literal $\forall(c:X\to G_2)$ in Γ there is a monomorphism from X to G', since we know that there is a monomorphism from X to G_1 and also from G_1 to G'. This means that every literal $\exists G'\in\mathcal{G}'$ and Γ satisfy the conditions of Proposition 8. Therefore, we have $(\bigvee_{G'\in\mathcal{G}'}\bigvee_{G\in\mathcal{G}'_G}\exists G)\vee\Gamma_1\in\bigcup_{k\geq 1}\mathcal{C}_k$.

  □

Let us now define the precedence relation mentioned above. The intuition is quite simple. $\exists G_1$ precedes $\exists G_2$ if G_1 is embedded in $\exists G_2$:

Definition 8 *For every pair of literals $\exists G_1,\exists G_2$, $\exists G_1\prec\exists G_2$ if there is a monomorphism $h_{G_1\prec G_2}:G_1\to G_2$.*

As said above, we use this precedence relation to build (or to find) models of the given specification. More precisely we use (possibly infinite) ascending sequences of basic constraints $\exists G_1\prec\cdots\prec\exists G_i\prec\ldots$ which are *saturated*, where intuitively a sequence is saturated if either it leads to a model of the given set of clauses, or if we know that the sequence cannot lead to a model (in this case we say that its last element is *closed*). Therefore, we define a *closed* literal as a literal that cannot be used for building a model of the given set of clauses.

Definition 9 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints, let $\mathcal{C}\Rightarrow\mathcal{C}_1\Rightarrow\cdots\Rightarrow\mathcal{C}_k\ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3) and let $BasPosLit(\bigcup_{k\geq 1}\mathcal{C}_k)$ be the set of all the basic positive literals occurring in clauses inferred in the refutation procedure. A literal $\exists G$ in $BasPosLit(\bigcup_{k\geq 1}\mathcal{C}_k)$ is* closed *if there is a strictly negative clause Γ in $\bigcup_{k\geq 1}\mathcal{C}_k$ such that $G\not\models\Gamma$. We also say that $\exists G$ is* open *if it is not closed.*

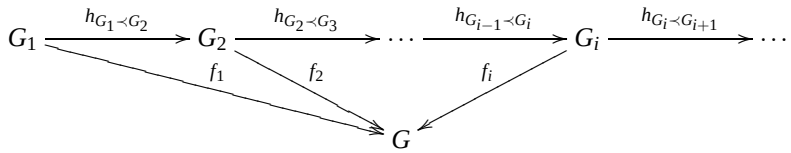
Following the intuitions above, a saturated sequence is a sequence of basic literals that approximate a model or, alternatively, that we have discovered that it is impossible that it leads to a model:

Definition 10 *An ascending sequence in BasPosLit*$(\bigcup_{k\geq 1} \mathcal{C}_k)$ $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ *is* saturated *if one of the following cases applies:*

- *the sequence is finite and its last element* $\exists G_k$ *satisfies that* G_k *is a model for* $\mathcal{C}$*, or*
- *the sequence is finite and its last element is closed, or*
- *the sequence is infinite and for every clause* Γ *in* $\bigcup_{k\geq 1} \mathcal{C}_k$ *there is a literal* L *in* Γ *such that:*
  (a) *if* $L = \neg\exists C$*, then for every* j *there is no monomorphism* $m : C \to G_j$
  (b) *if* $L = \exists C$*, there is a* j*, such that there is a monomorphism* $m : C \to G_j$
  (c) *If* $L = \forall(c : X \to C)$ *then for every* i *and every monomorphism* $m : X \to G_i$ *there is a* j*, with* $i < j$*, and a monomorphism* $h : C \to G_j$ *with* $h_{C_i \prec C_j} \circ m = h \circ c$*.*

The following lemma makes explicit in which sense an infinite saturated sequence provides successive approximations to a model of a given set of constraints:

Lemma 2 *Let* $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ *be an infinite saturated sequence in BasPosLit*$(\bigcup_{k\geq 1} \mathcal{C}_k)$ *for a fair refutation procedure* $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ *and let* G *be the colimit of the sequence:*



then G *is a model for the given set of clauses, i.e.* $G \models \mathcal{C}$*.*

Proof. Let Γ be any clause in $\mathcal{C}$. We have to prove that $G \models \Gamma$. Since the sequence is assumed to be saturated there should be a literal L in Γ such that the conditions (a), (b), or (c) in Def. 10 are satisfied. We consider each case separately:

(a) if $L = \neg\exists C$, then we know that for every j there is no monomorphism $m : C \to G_j$. But, according to Prop. 4, this means that there is no monomorphism $h : C \to G$. Therefore $G \models \neg\exists C$ and as a consequence $G \models \Gamma$.
(b) if $L = \exists C$, we know that there is a j, such that there is a monomorphism $m : C \to G_j$. But this means that there is a monomorphism $f_j \circ m : C \to G$. Therefore $G \models \exists C$ and as a consequence $G \models \Gamma$.
(c) If $L = \forall(c : X \to C)$ then we know that for every i and every monomorphism $m_0 : X \to G_i$ there is a j, with $i < j$, such that there is a monomorphism $h : C \to G_j$ with $h_{G_i \prec G_j} \circ m_0 = h \circ c$. Suppose that there is a monomorphism $m : X \to G$. This means, according to Prop. 4, that there exists an i such that there is a monomorphism $m' : X \to G_i$ such that $f_i \circ m' = m$. But this implies that there is a j, with $i < j$, such that there is a monomorphism $h : C \to G_j$ with $h_{G_i \prec G_j} \circ m' = h \circ c$. Hence, $f_j \circ h : C \to G$ and moreover $f_j \circ h \circ c = f_j \circ h_{G_i \prec G_j} \circ m' = f_i \circ m' = m$. Therefore, G satisfies $\forall c : X \to C$ and as a consequence $G \models \Gamma$.

□

The following two lemmas show that saturated sequences can be constructed using the I construction defined above. In particular, the first one shows how we can construct infinite sequences such that they are saturated if all its elements are open.

Lemma 3 *Let* $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ *be an infinite ascending sequence in BasPosLit*$(\bigcup_{k\geq 1} \mathcal{C}_k)$ *for a fair refutation procedure* $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ *such that, for every* j*,* $\exists G_j$ *is open and moreover* $h_{G_j \prec G_{j+1}} \in I(\exists G_j, \mathcal{C}_j)$*. Then,* $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ *is a saturated sequence.*

Proof. Let Γ be any clause in $\bigcup_{k\geq 1} \mathcal{C}_k$. More precisely, let us assume that $\Gamma \in \mathcal{C}_n$. We have to prove that there is a literal L in Γ such that:

(a) if $L = \neg\exists C$, then for every j there is no monomorphism $m : C \to G_j$
(b) if $L = \exists C$, there is a j, such that there is a monomorphism $m : C \to G_j$
(c) If $L = \forall(c : X \to C)$ then for every i and every monomorphism $m : X \to G_i$ there is a j, with $i < j$, and a monomorphism $f : C \to G_j$ with $h_{G_i \prec G_j} \circ m = f \circ c$.

Now, if there is an L in Γ that satisfies condition (a), then the proposition trivially holds. Otherwise, let us assume that there is a j such that for every negative literal $L = \neg\exists C$ in Γ, there is a monomorphism $m : C \to G_j$. Let $k = max(n, j)$. By Proposition 6, we know that for every $k' \geq k$ there should be a literal L in Γ such that:

- if $L = \exists C$, then there is a monomorphism $m : C \to G_{k'+1}$.
- If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_{k'}$ there is a monomorphism $f : C \to G_{k'+1}$ with $h_{G_{k'} \prec G_{k'+1}} \circ m = f \circ c$. Therefore, we just have to consider just the case when there is a monomorphism $m : X \to G_n$, with $n < k$: We know that $h_{G_n \prec G_k} \circ m : X \to G_k$ is a monomorphism then, by Proposition 6, there should be a monomorphism $f : C \to G_k$ with $h_{G_k \prec G_{k+1}} \circ h_{G_n \prec G_k} \circ m = f \circ c$. But $h_{G_k \prec G_{k+1}} \circ h_{G_n \prec G_k} = h_{G_n \prec G_{k+1}}$. Hence, $h_{G_n \prec G_k} \circ m = f \circ c$.

□

It may be noted that Lemma 3 (together with the rest of the results below), implicitly provides a procedure for building models of a given set of clauses. In particular, starting by the set of literals $\mathcal{L}_0$ consisting of the basic positive literals occurring in the given clauses, we build sets $\mathcal{L}_1, \ldots, \mathcal{L}_n, \ldots$ where each $\mathcal{L}_{i+1}$ is the set of basic literals in $I(\exists G, \mathcal{C}_j)$ which are not closed, and where $\exists G \in \mathcal{L}_i$. We can stop this construction if we find a literal $\exists G \in \mathcal{L}_n$ where G is already a model of the given specification.

The following lemma shows the existence of saturated sequence if the given set of clauses includes a basic positive constraint.

Lemma 4 *Given a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ if BasPosLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ is not empty then there is a saturated sequence in BasPosLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$.*

Proof. Let us suppose that there is a literal $\exists G$ in *BasPosLit*$(\bigcup_{k\geq 1} \mathcal{C}_k)$. We define a sequence $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ in *BasPosLit*$(\bigcup_{k\geq 1} \mathcal{C}_k)$ as follows:

- $G_1 = G$.
- If $h : G_j \to G'$ is a monomorphism such that $h \in I(\exists G_j, \mathcal{C}_j)$, then we define $G_{j+1} = G'$.

Now, we have to prove that this sequence is saturated. We consider three cases:

- The sequence is finite because $I(\exists G_j, \mathcal{C}_j)$ is the empty set. This means that $G_j \not\models \Gamma$ for some strictly negative clause $\Gamma \in \mathcal{C}_j$. But this means that Γ is closed and, as a consequence, the sequence $\exists G_1 \prec \cdots \prec \exists G_j$ is saturated.
- The sequence is finite, because $id_{G_j} \in I(\exists G_j, \mathcal{C})$. Then, according to Proposition 7, this means that G_j is a model for $\mathcal{C}_j$ and, hence, for $\mathcal{C}$. As a consequence, the sequence $\exists G_1 \prec \cdots \prec \exists G_j$ is saturated.
- Otherwise, the sequence is infinite and, for every j, $\exists G_j$ is open. Then, by Lemma 3, the sequence $\exists G_1 \prec \cdots \prec \exists G_j \prec \ldots$ is saturated.

□

The last result that we need, before proving completeness for our inference rules, shows that if all saturated sequences end in a closed literal and if the given set of constraints includes a clause consisting only of basic positive literals then we can infer a clause consisting only of closed literals.

Lemma 5 *Let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3) such that $\mathcal{C}$ includes a clause Γ consisting only of basic positive literals. If every saturated sequence in BasPosLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ is finite and its last element is a closed literal then there is a clause Γ' in $\bigcup_{k\geq 1} \mathcal{C}_k$ consisting only of closed literals.*

Proof. We define inductively the sequence of clauses $\Gamma_1, \ldots, \Gamma_n, \ldots$ where:

- $\Gamma_1 = \Gamma$.
- $\Gamma_{n+1} = (\bigvee_{G \in \mathcal{G}_{n+1}} \exists G)$, where $\mathcal{G}_{n+1} = \{G \mid$ there is a literal $\exists G' \in \Gamma_n$ with $(h : G' \to G) \in I(\exists G', \mathcal{C}_n\}$

Now, we know that every set of clauses $\mathcal{C}_k$ is finite and this implies that, for every literal $\exists C$ in Γ_k, $I(\exists C, \mathcal{C}_k)$ is also finite. As a consequence, if for every i there is an open literal included in Γ_i then this means that there should be an infinite sequence of open literals $\exists G_1 \prec \cdots \prec \exists G_n \prec \ldots$ where each $G_n \in \Gamma_n$ and $h_{G_n \prec G_{n+1}} \in I(\exists G_n, \mathcal{C}_n)$. But by Lemma 3 this sequence would be saturated against our original assumption. Therefore, there should exist an i where all the literals in Γ_i are closed. So it is enough to define $\Gamma' = \Gamma_i$ □

Lemma 6 ***(Completeness)*** *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints, let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3). If $\mathcal{C}$ is unsatisfiable then there is a j such that the empty clause is in $\mathcal{C}_j$.*

Proof. Suppose that the empty clause is not in $\mathcal{C}_j$ for any j. We have to show the existence of a graph G such that $G \models \mathcal{C}$. We consider four cases:

1. There is no clause Γ in $\mathcal{C}$ consisting only of basic positive literals. This means that every clause Γ includes a negative literal $\neg\exists C$ or a non-basic literal $\forall(c : X \to C)$, where X is not empty. In this case, the empty graph would satisfy all these atomic and negative literals and, as a consequence, would be a model for $\mathcal{C}$.
2. Otherwise, we have a clause Γ in $\mathcal{C}$ consisting only of basic positive literals. Then, by Lemma 4, we know that there exist at least one saturated sequence in $\mathit{BasPosLit}(\bigcup_{k\geq 1} \mathcal{C}_k)$. By Def. 10, we have the following cases:
   (a) Every saturated sequence in $\mathit{BasPosLit}(\bigcup_{k\geq 1} \mathcal{C}_k)$ is finite and its last element is a closed literal. We may see that this case is not possible. Let $\exists G \vee \Gamma$ be a minimal clause in $\bigcup_{k\geq 1} \mathcal{C}_k$ consisting only of closed literals (according to Lemma 5 we know that such clause must exist and, according to our assumption, it must no be empty). Since we are assuming that $\exists G$ is closed then there should exist a clause $\neg\exists C_1 \vee \cdots \vee \neg\exists C_n$ in $\bigcup_{k\geq 1} \mathcal{C}_k$ such that for every i there is a monomorphism $m_i : C_i \to G$. Using rule (R1) we can infer $\Gamma \vee \neg\exists C_2 \vee \cdots \vee \neg\exists C_n$. Then, using again rule (R1) with this clause and the clause $\exists G \vee \Gamma$, we can infer $\Gamma \vee \Gamma \vee \neg\exists C_3 \vee \cdots \vee \neg\exists C_n = \Gamma \vee \neg\exists C_3 \vee \cdots \vee \neg\exists C_n$. Then, applying repeatedly rule (R1) in a similar way, we would finally infer Γ, against the assumption that $\exists G \vee \Gamma$ was minimal.
   (b) There is a finite saturated sequence in $\mathit{BasPosLit}(\bigcup_{k\geq 1} \mathcal{C}_k)$ whose last element is $\exists G$. Then $G \models \mathcal{C}$.
   (c) There is an infinite saturated sequence $\exists G_1 \prec \cdots \prec \exists G_i \prec \ldots$ in $\mathit{BasPosLit}(\bigcup_{k\geq 1} \mathcal{C}_k)$. Then, according to Lemma 2, its colimit is a model for the given set of clauses.

□

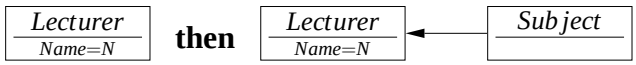
As a consequence of Lemmas 1 and 6 , we have:

Theorem 1 ***(Soundness and Completeness)*** *Let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over a set of basic constraints and positive atomic constraints$\mathcal{C}$, based on the rules (R1), (R2), and (R3). Then, $\mathcal{C}$ is unsatisfiable if and only if there is a j such that the empty clause is in $\mathcal{C}_j$.*

Example 3 *In our running example, the two models of the given set of constraints $\mathcal{C}$ (i.e. the two graphs in (10)) would be built in $I(\exists CS1, \{(2),(3),(4),(5),(6)\})$, where $\exists CS1$ is the only literal in constraint (1). More precisely, $I(\exists CS1, \{(2),(3),(4),(5),(6)\}) = I(\exists CS1, \{(2),(3)\})$, since (4), (5) and (6) do not include any positive literal. Then, $I(\exists CS1,(2))$ would consist of the inclusion from the graph $CS1$ to the graph in (8), let us call it $CS12$. And $I(\exists CS12,(3))$ would consist of the inclusions from the graph $CS12$ to the graphs in (9). Now two of these four graphs do not satisfy clause (4), which means that they are closed. The other two graphs, as said above are models of the given set of constraints.*

4. Atomic Constraints

The approach used in the previous section and the results obtained cannot be directly extended in an obvious way to deal with the general case of specifications including also negative atomic constraints. Let us see the problem. In the previous section, the idea of the approach followed was that we had two kinds of rules. Rules (R2) and (R3) were seen as rules to *build* models of the given positive constraints, while the rule (R1) was used to *discard* models not satisfying the negative constraints. This is the idea of the proof of Lemma 6. Now, suppose that our specification includes the constraint $\neg\forall(g : X \to C)$. This constraint, although it is a negative constraint, can be considered similar to a positive constraint in the sense that, if we have a graph G that does not satisfy it, we can build a new graph G' that satisfies the constraint by extending G with some new nodes and edges so that it includes a copy of X (without including its extension C). For instance, given the constraint:

(7) $\neg$ **if** 

If a graph G does not satisfy the constraint because all lecturer nodes are linked to some subject node, then we can add to G a new lecturer node and the resulting graph will now satisfy the constraint. This intuition suggests that the rule below could be what is needed to deal with negative atomic constraints:

$$\frac{\exists C_1 \vee \Gamma_1 \quad \neg\forall(g : X \to C) \vee \Gamma_2}{(\bigvee_{G \in \mathcal{G}} \exists G) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R4)}$$

where $\mathcal{G} = \{G \mid \langle f_1 : C_1 \to G \leftarrow X : f_2 \rangle \in (C_1 \otimes X)$ such that there is no $m : C \to G$ with $f_2 = m \circ g\}$.

The above rule can be proven sound but, unfortunately, we can see very easily that rules (R1) - (R4) are incomplete. It is enough to consider a specification consisting of the constraints $\forall(g : X \to C)$ and $\neg\forall(g : X \to C)$. The specification is trivially unsatisfiable. However, we cannot derive the empty clause using rules (R1) - (R4). Let us see what would fail in the completeness proof, if we try to do it along similar lines as the proof for Lemma 6.

Suppose that we have a graph that does not satisfy the negative constraint (for instance the empty graph) then, using rule (R4), we could build a graph that satisfies it, in this case the graph X. But this graph does not satisfy the constraint $\forall(g : X \to C)$. Then, according to rule (R3), we can now build (among others) the graph C that satisfies the positive constraint. Unfortunately, C now does not satisfy the constraint $\neg\forall(g : X \to C)$. That is, the main difference between the current situation and the proof of Lemma 6 is that in the latter case if we have that $G_1 \prec G_2$ then G_2 could be seen closer (a better approximation) to a graph that satisfies all the positive constraints. However, in the former case, G_1 may be satisfying the constraint $\neg\forall(g : X \to C)$ while none of its successors satisfies that constraint.

The idea of the proposed solution to avoid this problem is, first, to annotate the basic atoms in the clauses with information about the negative constraints that have been used to infer that clause. And, then, to use this information so that, when doing a new inference, the basic atoms included in the resulting clause still satisfy the negative constraints included in the annotation. These annotations are called *contexts*, and the annotated constraints are called *contextual constraints*. More precisely, given a constraint $\exists C$, a context for this constraint is a set of negative atomic constraints $\neg\forall(g : X \to C_1)$ such that X is included in C. Actually, we assume that C has to satisfy this negative constraint. However, as usual, we need to know not only that X is a subgraph of C, but also to identify the specific instance of X that cannot be extended to C_1. For this reason we consider that a context is a finite set of negative atomic constraints together with monomorphisms binding the conditional part of each constraint to the corresponding literal. Below, in the completeness proof, we will see in more detail the use of these contexts.

Definition 11 ***(Contextual Constraints)*** *A contextual constraint $\exists C[\mathcal{Q}]$ is a pair consisting of a basic constraint, $\exists C$, and a set $\mathcal{Q}$ consisting of pairs $\langle \neg\forall(g : X \to C_1), h : X \to C \rangle$ where $\neg\forall(g : X \to C_1)$ is a negative atomic constraint and h is a monomorphism. A contextual constraint $\exists C[\mathcal{Q}]$ is consistent if for each pair $\langle \neg\forall(g : X \to C_1), h : X \to C \rangle$ in $\mathcal{Q}$ there is no monomorphism $h' : C_1 \to C$ such that $h = h' \circ g$.*

A constraint $\exists C$ without a context is considered to be annotated by the empty context. Now, we have to define satisfaction for this kind of contextual constraints. The idea is that a graph satisfies a contextual constraint $\exists C[\mathcal{Q}]$ if it satisfies $\exists C$ and all the constraints in its context:

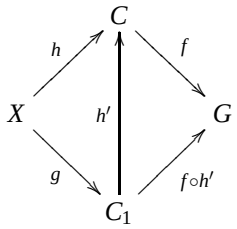
Definition 12 ***(Satisfaction of Contextual Constraints)*** *A graph G satisfies a contextual constraint $\exists C[\mathcal{Q}]$ via a monomorphism $f : C \to G$, written $G \models_f \exists C[\mathcal{Q}]$, if for every pair $\langle \neg\forall(g : X \to C_1), h : X \to C \rangle \in \mathcal{Q}$ there is no monomorphism $h' : C_1 \to G$ such that $f \circ h = h' \circ g$. G satisfies $\exists C[\mathcal{Q}]$, written $G \models \exists C[\mathcal{Q}]$, if there is a monomorphism $f : C \to G$ such that $G \models_f \exists C[\mathcal{Q}]$.*

Inconsistent contextual constraints are not satisfied by any graph:

Fact 2 *If $\exists C[\mathcal{Q}]$ is not consistent then for every graph G, $G \not\models \exists C[\mathcal{Q}]$.*

Proof. Suppose that $\exists C[\mathcal{Q}]$ is inconsistent, i.e. there is a pair $\langle \neg\forall(g : X \to C_1), h : X \to C \rangle$ in $\mathcal{Q}$ and there is a monomorphism $h' : C_1 \to C$ such that $h = h' \circ g$.

If there is a morphism $f : C \to G$ then according to the diagram below:



we have that $f \circ h' \circ g = f \circ h$, which means that $G \not\models \exists C[\mathcal{Q}]$. □

Finally, in some inference rules, given a contextual constraint $\exists C[\mathcal{Q}]$ and a monomorphism $f : C \to G$, we need to be able to build a contextual constraint whose left-hand side is $\exists G$ and whose context includes the same negative constraints as $[\mathcal{Q}]$. In order to do this we need to define the new binding to G of the negative constraints in $[\mathcal{Q}]$:

Definition 13 *Given a contextual constraint $\exists C[\mathcal{Q}]$ and a monomorphism $f : C \to G$, we define the context $f\langle\mathcal{Q}\rangle$ as the set $\{\langle\neg\forall(g : X \to C_1), f \circ h : X \to G\rangle \mid \langle\neg\forall(g : X \to C_1), h : X \to C\rangle \in \mathcal{Q}\}$.*

In this case, satisfiability is based on five rules. The first three rules are a reformulation (in terms of contextual constraints) of the rules defined in the previous sections. The fourth rule is a similar reformulation of the rule stated above. In addition , there is a new rule that states that contextual constraints that are not consistent can be deleted from a clause. The five rules are:

$$\frac{\exists C_1[\mathcal{Q}_1] \vee \Gamma_1 \quad \neg\exists C_2 \vee \Gamma_2}{\Gamma_1 \vee \Gamma_2} \quad \textbf{(R1')}$$

if there exists a monomorphism $m : C_2 \to C_1$

$$\frac{\exists C_1[\mathcal{Q}_1] \vee \Gamma_1 \quad \exists C_2[\mathcal{Q}_2] \vee \Gamma_2}{(\bigvee_{G \in \mathcal{G}} \exists G[f_1\langle\mathcal{Q}_1\rangle \cup f_2\langle\mathcal{Q}_2\rangle]) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R2')}$$

where $\mathcal{G} = \{G \mid \langle f_1 : C_1 \to G \leftarrow C_2 : f_2\rangle \in (C_1 \otimes C_2)\}$.

$$\frac{\exists C_1[\mathcal{Q}] \vee \Gamma_1 \quad \forall(c : X \to C_2) \vee \Gamma_2}{(\bigvee_{G \in \mathcal{G}} \exists G[f_1\langle\mathcal{Q}\rangle]) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R3')}$$

if there is a monomorphism $m : X \to C_1$ and $\mathcal{G} = \{G \mid \langle f_1 : C_1 \to G \leftarrow C_2 : f_2\rangle \in (C_1 \otimes C_2)$ such that $f_1 \circ m = f_2 \circ c\}$.

$$\frac{\exists C_1[\mathcal{Q}_1] \vee \Gamma_1 \quad \neg\forall(g : X \to C_2) \vee \Gamma_2}{(\bigvee_{\langle G,\mathcal{Q}\rangle \in \mathcal{G}} \exists G[\mathcal{Q}]) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R4')}$$

where $\mathcal{G} = \{\langle G, f_1\langle\mathcal{Q}_1\rangle \cup \{\langle\neg\forall(g : X \to C_2), f_2\rangle\}\rangle \mid \langle f_1 : C_1 \to G \leftarrow X : f_2\rangle \in (C_1 \otimes X)\}$.

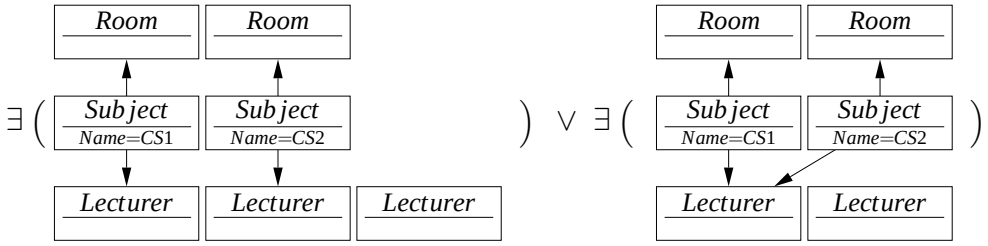
$$\frac{\exists C[\mathcal{Q}] \vee \Gamma}{\Gamma} \quad \textbf{(R5)}$$

if $\exists C[\mathcal{Q}]$ is not consistent.

We may see that (R4') is very similar to (R2'). The reason is that, as discussed above, a negative atomic constraint $\neg\forall(c : X \rightarrow C_2)$ (partly) specifies that there must be a copy of X in the given graph, as it happens with the constraint $\exists X$. The main difference to the rule (R2') is that, in (R4'), the negative constraint is added to the context of the new constraints introduced in the clause inferred by the rule. As said above, the fifth rule just states that inconsistent contextual constraints can be deleted from clauses, since they cannot be satisfied by any graph.

Example 4 *Let us consider all the constraints and clauses from Examples 1 and 2. If we apply twice the rule (R4') on clauses (10) and (7) then we would infer the following clause:*

(11)

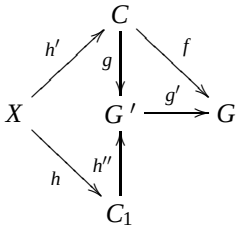


where the context associated to each literal (not displayed above) would consist of constraint (7) together with a monomorphism mapping the Lecturer node in the condition part of (7) to the Lecturer node which is disconnected in each of the graphs. Again, no useful new inferences can be applied, and the two graphs occurring in clause (11) are (minimal) models of the set of constraints.

Now, with this new formulation, again we are able to show soundness and completeness of our inference rules. In particular, the proofs of soundness for rules (R1')-(R3') are a straightforward extension of the proofs for rules (R1)-(R3). The only difference is that we have to take into account the contexts. Anyhow, before showing the soundness and completeness of the new calculus let us prove a proposition about satisfaction of contextual constraints.

Proposition 10 *Let $\exists C[Q]$ be a consistent contextual literal, let G be a graph such that $G \models_f \exists C[Q]$, and let $g : C \rightarrow G'$ and $g' : G' \rightarrow G$ be monomorphisms such that $f = g' \circ g$. Then, $G'[g\langle Q\rangle]$ is consistent and $G \models_{g'} \exists G'[g\langle Q\rangle]$.*

Proof. Let $\langle \neg\forall(h : X \rightarrow C_1), h' : X \rightarrow C\rangle \in Q$, on one hand we have to prove that there is no monomorphism $h'' : C_1 \rightarrow G'$ such that $h'' \circ h = g \circ h'$. However the existence of h'' would imply that we have the monomorphism $g' \circ h'' : C_1 \rightarrow G$ satisfying that $g' \circ h'' \circ h = g' \circ g \circ h' = f \circ h'$ against the hypothesis that $G \models_f \exists C[Q]$. Therefore $G'[g\langle Q\rangle]$ is consistent.



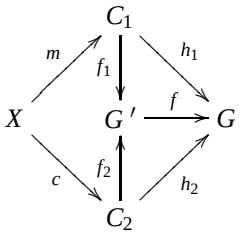
On the other hand, we have to prove that there is no monomorphism $f' : C_1 \rightarrow G$ such that $f' \circ h = g' \circ g \circ h'$, but this is straightforward since $f = g' \circ g$ and $G \models_f \exists C[Q]$. Therefore $G \models_{g'} \exists G'[g\langle Q\rangle]]$. □

Lemma 7 ***(Soundness of the rules)*** *The rules (R1'), (R2'), (R3'), (R4'), and (R5) are sound.*

Proof. The proofs for the rules (R1'-R3') are similar to the proofs for the rules (R1-R3). Below, in addition to the proofs for the new rules (R4') and (R5), we just present the proof for the rule (R3') to show the (small) difference to the proof of the corresponding rule (R3).

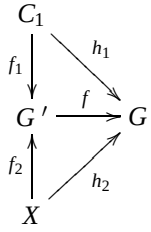
(R3') Suppose that $G \models \exists C_1[Q] \vee \Gamma_1$, $G \models \forall(c : X \rightarrow C_2) \vee \Gamma_2$, and there is a monomorphism $m : X \rightarrow C_1$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Suppose that $G \models_{h_1} \exists C_1[Q]$, for some monomorphism $h_1 : C_1 \rightarrow G$. In addition, we also have that that there is a monomorphism $h_2 : C_2 \rightarrow G$ such that $h_1 \circ m = h_2 \circ c$, since $G \models \forall(c : X \rightarrow C_2)$. As a

consequence, by Prop. 2 there is a factorization:



where $f_1 : C_1 \to G'$ and $f_2 : C_2 \to G'$ are jointly surjective monomorphisms and f is injective. Therefore, G' is in the set $\mathcal{G}$ defined in the rule. Finally, according to Prop. 10, we have that $G \models_f \exists G'[f_1\langle Q\rangle]$ which means that $G \models (\bigvee_{G' \in \mathcal{G}} \exists G'[f_1\langle Q\rangle]) \vee \Gamma$

(R4') Similarly, suppose that $G \models \exists C_1[Q_1] \vee \Gamma_1$ and $G \models \neg\forall(c : X \to C_2) \vee \Gamma_2$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Suppose that $G \models_{h_1} \exists C_1[Q_1]$, for some monomorphism $h_1 : C_1 \to G$, and that there is a monomorphism $h_2 : X \to G$ such that there is no monomorphism $h' : C_2 \to G$ such that $h_2 = h' \circ c$, i.e. $G \models \neg\forall(c : X \to C_2)$. As a consequence, by Prop. 1 there is a factorization:



where $f_1 : C_1 \to G'$ and $f_2 : X \to G'$ are jointly surjective monomorphisms and f is injective. Note that there is no monomorphism $h : C_2 \to G'$ such that $f_2 = h \circ c$, since $f \circ f_2 = h_2$ and this would mean that $h_2 = f \circ h \circ c$ violating the above condition. Hence, we have that $G' \in \{G'' \mid \langle f_1 : C_1 \to G'' \leftarrow C_2 : f_2\rangle \in (C_1 \otimes C_2)\}$. Finally, on one hand, according to proposition 10, we have that $G \models_f \exists G'[f_1\langle Q_1\rangle]$ and, on the other hand, we can show that $G \models_f G'[\{\langle\neg\forall(g : X \to C_2), f_2\rangle\}]$. The reason is that if $h' : C_2 \to G$ is a monomorphism such that $f \circ f_2 = h' \circ c$, we would have that $G \models \neg\forall(c : X \to C_2)$ would not hold, since we would have $h_2 = f \circ f_2 = h' \circ c$. Altogether, this means that $G \models \exists G'[Q])$ for $Q = f_1\langle Q_1\rangle \cup \{\langle\neg\forall(g : X \to C_2), f_2\rangle\}$. As a consequence, $G \models (\bigvee_{\langle G',Q\rangle \in \mathcal{G}} \exists G[Q])$, for $\mathcal{G} = \{\langle G', f_1\langle Q_1\rangle \cup \{\langle\neg\forall(g : X \to C_2), f_2\rangle\}\rangle \mid \langle f_1 : C_1 \to G \leftarrow X : f_2\rangle \in (C_1 \otimes X)\}$.

(R5) Suppose that $G \models \exists C[Q] \vee \Gamma$. By Fact 2 we know that $G \not\models \exists C[Q]$. Therefore, $G \models \Gamma$. □

The proof of completeness in this case is very similar to the previous completeness proof. The main difference is in the key role played by the contexts. The idea in the previous proof was to consider sequences of constraints $\exists C_1 \prec \cdots \prec \exists C_i \prec \ldots$, where every C_i is included in C_{i+1}, that could be seen as the construction of a model for $\mathcal{C}$ if the empty clause was never inferred. In particular, these sequences were associated to the given inferences. Moreover, an important property in that proof is that it was assumed that every graph in these sequences would satisfy all the strictly negative clauses in $\bigcup_{k\geq 1} \mathcal{C}_k$. In particular, given a graph C_i, if a possible successor C_{i+1} does not satisfy a strictly negative clause $\neg\exists C_1 \vee \cdots \vee \neg\exists C_n$ then we know that a sequence $\emptyset \prec C_1 \prec \cdots \prec C_i \prec C_{i+1} \prec \ldots$ would never yield a model of $\mathcal{C}$. The reason is that any graph including C_i will neither satisfy $\neg\exists C$.

In the current case, as discussed above, negative atomic constraints are treated in a similar way to basic positive constraints. If a graph C_i does not satisfy the constraint $\neg\forall(g : X \to C)$ then we may build C_{i+1} including a copy of X (but not of its extension C) applying the fourth rule. This means that C_{i+1} now satisfies that constraint. However, in this situation if we do not use contexts, it would be impossible to say if this sequence, in the limit (or, rather, in the colimit) would yield a model of $\mathcal{C}$ and, especially, if it would satisfy that constraint. The reason is that C_{i+2} may include a copy of C as an extension of the instance of X included in C_{i+1}.

The use of contexts solves this problem. In particular, if $C_i[Q]$ does not satisfy a constraint $\neg\forall(g : X \to C)$ in its context Q then no larger graph would satisfy it. Then, in a similar manner as in the previous completeness proof, we

can define sequences $C_0[\emptyset] \prec C_1[\mathcal{Q}_1] \prec \cdots \prec C_i[\mathcal{Q}_i] \prec C_{i+1}[\mathcal{Q}_{i+1}] \prec \ldots$, where each C_i satisfies all the strictly negative clauses and all the negative constraints in $\mathcal{Q}_i$. Then, saturation of the sequences ensures that for every sequence there is an i such that $\mathcal{Q}_i$ includes all the negative atomic constraints in $\mathcal{C}$. This ensures that a saturated sequence will yield a model of $\mathcal{C}$, provided that the empty clause cannot be inferred from $\mathcal{C}$.

As in the previous completeness proof, we have to provide some auxiliary definitions and results. These definitions and results are in most cases essentially equivalent to the corresponding ones in the previous section. In particular, in some cases the only difference would be that the given basic constraints will have a context. In some other cases, we will explicitly have to deal with the new kind of constraints considered (i.e. non-basic negative constraints). For this reason, we will omit the proof of these auxiliary results when they are essentially identical to the corresponding proof in Section 3, or we will just show the proof for the case of the new constraints, when this is the only difference.

We start defining the construction I. In this case, the result of $I(\exists G_1[\mathcal{Q}_1], L)$ is not the set of possible graphs (actually monomorphisms from G_1 to these graphs) that we can infer from $\exists G_1[\mathcal{Q}_1]$ and L, but also the resulting contexts:

Definition 14 *Let $\exists G_1[\mathcal{Q}_1]$ be a contextual literal and L a positive literal or a negative non-basic constraint. We define the set of monomorphisms $I(\exists G_1[\mathcal{Q}_1], L)$ by cases:*

- *If L is a basic contextual literal, $L = \exists G_2[\mathcal{Q}_2]$, then $I(\exists G_1[\mathcal{Q}_1], L) = \{\langle f_1, \mathcal{Q}\rangle \mid \langle f_1 : G_1 \to G \leftarrow G_2 : f_2\rangle \in (G_1 \otimes G_2)\}$, where $\mathcal{Q} = f_1\langle\mathcal{Q}_1\rangle \cup f_2\langle\mathcal{Q}_2\rangle$.*
- *If L is a positive atomic literal, $L = \forall(c : X \to G_2)$, and H is the set of all monomorphisms from X to G_1 then, $I(\exists G_1[\mathcal{Q}_1], L) = I^*(\exists G_1[\mathcal{Q}_1], H)$, where $I^*(\exists G_1[\mathcal{Q}_1], H)$ is defined inductively:*
  - *If $H = \emptyset$ then $I^*(\exists G_1[\mathcal{Q}_1], H) = \{\langle id_{G_1}, \mathcal{Q}_1\rangle\}$.*
  - *If $H = \{f : X \to G_1\} \cup H'$ then $I^*(\exists G_1[\mathcal{Q}_1], H) = \{\langle h' \circ h, h'\langle\mathcal{Q}\rangle\rangle \mid \langle h : G_1 \to G, \mathcal{Q}\rangle \in I^*(\exists G_1[\mathcal{Q}_1], H'), \langle h' : G \to G' \leftarrow G_2 : f_2\rangle \in (G \otimes G_2)$ such that $f_2 \circ c = h' \circ h \circ f\}$.*
- *If L is a negative non-basic constraint, $L = \neg\forall(c : X \to G_2)$, then $I(\exists G_1[\mathcal{Q}_1], L) = \{\langle f_1, \mathcal{Q}\rangle \mid \langle f_1 : G_1 \to G \leftarrow X : f_2\rangle \in (G_1 \otimes X)\}$, where $\mathcal{Q} = f_1\langle\mathcal{Q}_1\rangle \cup \{\langle\forall(c : X \to G_2), f_2\rangle\}$.*

The definition of I is extended to clauses and sets of clauses as in the previous section:

Definition 15 *Let $\exists G_1[\mathcal{Q}_1]$ be a contextual literal and Γ be a clause. We define the set of monomorphisms $I(\exists G_1[\mathcal{Q}_1], \Gamma)$ inductively:*

- *If Γ is the empty clause, then $I(\exists G_1[\mathcal{Q}_1], \Gamma) = \emptyset$.*
- *If $\Gamma = L \vee \Gamma'$, where L is a negative basic literal, then $I(\exists G_1[\mathcal{Q}_1], \Gamma) = I(\exists G_1[\mathcal{Q}_1], \Gamma')$.*
- *If $\Gamma = L \vee \Gamma'$, where L is a positive literal or a negative non-basic literal, then $I(\exists G_1[\mathcal{Q}_1], \Gamma) = I(\exists G_1[\mathcal{Q}_1], L) \cup I(\exists G_1[\mathcal{Q}_1], \Gamma')$.*

If $\exists G_1[\mathcal{Q}_1]$ is a contextual literal and $\mathcal{C}$ is a set of clauses, the set of monomorphisms $I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$ is defined inductively:

- *If $\mathcal{C}$ is the empty set, then $I(\exists G_1[\mathcal{Q}_1], \mathcal{C}) = \{\langle id_{G_1}, \mathcal{Q}_1\rangle\}$.*
- *If $\mathcal{C} = \{\Gamma\} \cup \mathcal{C}'$, and Γ includes a negative basic literal, $\neg\exists G$ such that $G_1 \models \neg\exists G$, or Γ includes a positive atomic literal, $\forall(c : X \to G_2)$ such that there is no monomorphism $h : X \to G_1$ then $I(\exists G_1[\mathcal{Q}_1], \mathcal{C}) = I(\exists G_1[\mathcal{Q}_1], \mathcal{C}')$.*
- *Otherwise,*

$$I(\exists G_1[\mathcal{Q}_1], \{\Gamma\} \cup \mathcal{C}') = \{\langle g \circ h, \mathcal{Q}\rangle \mid \langle g, \mathcal{Q}\rangle \in I(\exists G[\mathcal{Q}'], \Gamma), \langle h : G_1 \to G, \mathcal{Q}'\rangle \in I(\exists G_1[\mathcal{Q}_1], \mathcal{C}')\}$$

The function I is monotonic with respect to the context part:

Proposition 11

- *If L is not a basic negative literal and $\langle h, \mathcal{Q}\rangle \in I(\exists G_1[\mathcal{Q}_1], L)$ then $h\langle\mathcal{Q}_1\rangle \subseteq \mathcal{Q}$.*
- *For any clause Γ, if $\langle h, \mathcal{Q}\rangle \in I(\exists G_1[\mathcal{Q}_1], \Gamma)$ then $h\langle\mathcal{Q}_1\rangle \subseteq \mathcal{Q}$.*
- *For any set of clauses $\mathcal{C}$, if $\langle h, \mathcal{Q}\rangle \in I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$ then $h\langle\mathcal{Q}_1\rangle \subseteq \mathcal{Q}$.*

Proof. Straighforward by the definition of I. □

The following proposition is almost identical to Prop. 5 since here the contexts do not play any role. The main difference is that now we also consider the case of non-basic negative constraints.

Proposition 12 *Let Γ be a clause and let $h : G_1 \to G_2$ be a monomorphism such that $\langle h, \mathcal{Q} \rangle \in I(\exists G_1[\mathcal{Q}_1], \Gamma)$, for some set of contexts $\mathcal{Q}$ then there is a literal L in Γ such that:*

- *if $L = \neg\exists C$, then there is no monomorphism $m : C \to G_1$.*
- *if $L = \exists C[\mathcal{Q}]$, then there is a monomorphism $m : C \to G_2$.*
- *If $L = \forall(c : X \to C)$ then for every monomorphism $f : X \to G_1$ there is a monomorphism $g : C \to G_2$ with $h \circ f = g \circ c$.*
- *If $L = \neg\forall(c : X \to C)$, then $\langle \neg\forall(c : X \to C), m \rangle \in \mathcal{Q}$ for some monomorphism $m : X \to G_2$.*

Proof. If Γ_0 is the subset of Γ including all its positive literals and all its non-basic negative constraints then, we know that $I(\exists G_1[\mathcal{Q}_1], \{\Gamma\}) = \bigcup_{L \in \Gamma_0} I(\exists G_1[\mathcal{Q}_1], L])$. Now, if Γ does not include any positive literal nor a negative non basic constraint (i.e. Γ is the empty clause or Γ includes only negative basic literals), then the proposition trivially holds, since by definition $I(\exists G_1[\mathcal{Q}_1], \Gamma)$ is empty. So let us assume that Γ includes some literal which is not a negative basic constraint and, moreover, let us assume that $\langle (h : G_1 \to G_2), \mathcal{Q} \rangle \in I(\exists G_1[\mathcal{Q}_1], \neg\forall(c : X \to C))$ for some literal $\neg\forall(c : X \to C)$ in Γ, since the other possible cases were proved in Prop. 5 and the current proof would be essentially identical. But this case is quite straightforward since, by definition, $I(\exists G_1[\mathcal{Q}_1], \neg\forall(c : X \to C)) = \{\langle h, \mathcal{Q} \rangle \mid \langle h : G_1 \to G \leftarrow X : m \rangle \in (G_1 \otimes X)\}$, where $\mathcal{Q} = h\langle \mathcal{Q}_1 \rangle \cup \{\langle \forall(c : X \to G_2), m \rangle\}$.

□

The extension of the above proposition to a set of clauses $\mathcal{C}$ is again almost identical to Prop. 6, except for the case when the literal L is a non-basic negative constraint.

Proposition 13 *Let $\mathcal{C}$ be a set of clauses consisting of basic constraints and positive atomic constraints and let $h : G_1 \to G_2$ be a monomorphism such that $\langle h, \mathcal{Q} \rangle \in I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$, then for every clause Γ in $\mathcal{C}$ there is a literal L in Γ such that:*

- *if $L = \neg\exists C$, then there is no monomorphism $m : C \to G_1$.*
- *if $L = \exists C[\mathcal{Q}]$, then there is a monomorphism $m : C \to G_2$.*
- *If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_1$ there is a monomorphism $f : C \to G_2$ with $h \circ m = f \circ c$.*
- *If $L = \neg\forall(c : X \to C)$, then $\langle \neg\forall(c : X \to C), m \rangle \in \mathcal{Q}$ for some monomorphism $m : X \to G_2$.*

Proof. Let $\langle h, \mathcal{Q} \rangle \in I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$. By induction on $\mathcal{C}$, following the definition of $I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$:

- If $\mathcal{C}$ is the empty set, then the proposition trivially holds.
- Otherwise, by induction, we know that if $\langle h' : G_1 \to G_2', \mathcal{Q}' \rangle \in I(\exists G_1[\mathcal{Q}_1], \mathcal{C}')$ every Γ' in $\mathcal{C}'$ satisfies the proposition with respect to h'. Therefore, if $h = g \circ h'$, with $\langle g : G_2' \to G_2, \mathcal{Q}' \rangle \in I(\exists G_2'[\mathcal{Q}'], \Gamma)$, on one hand we have to prove that every Γ' in $\mathcal{C}'$ satisfies the proposition with respect to $g \circ h'$ and, on the other, that Γ also satisfies the proposition with respect to $g \circ h'$.
  Given a clause Γ' in $\mathcal{C}'$, by induction, we know that there is a a literal L in Γ such that one of the following cases hold:
  - if $L = \neg\exists C$, the case is trivial.
  - if $L = \exists C[\mathcal{Q}'']$, then there is a monomorphism $m : C \to G_2'$. But this means that $g \circ m : C \to G_2'$
  - If $L = \forall(c : X \to C)$ then for every monomorphism $m : X \to G_1$ there is a monomorphism $f : C \to G_2'$ with $h' \circ m = f \circ c$. But this means that there is a monomorphism $g \circ f : C \to G_2$. Moreover, $g \circ f \circ c = g \circ h' \circ m = h \circ m$
  - If $L = \neg\forall(c : X \to C)$ then $\langle \neg\forall(c : X \to C), m' \rangle \in \mathcal{Q}'$ but this means that $\langle \neg\forall(c : X \to C), g \circ m' \rangle \in \mathcal{Q}$.

  Now, in the case of the clause Γ the proof is just a direct consequence of Prop. 12.

□

The aim of the following two propositions, like in the case of Propositions 8 and 9, is to show that if we have a literal $\exists G_1[\mathcal{Q}_1]$ in a clause Γ, then we can infer the clause resulting from replacing that literal by a disjunction of the literals that, in some sense can be considered included in $I(\exists G_1[\mathcal{Q}_1], \mathcal{C})$. The proof of Proposition 14 is very similar to

the proof of Proposition 8. The main difference, in addition to dealing with the contexts involved, is that in Proposition 14 we have to explicitly deal with the case of non basic constraints.

Proposition 14 *Let $\mathcal{C}$ be a set of clauses, let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1'), (R2'), (R3'), (R4'), and (R5), and let $\exists G_1[Q_1] \vee \Gamma_1$ and Γ_2 be two non-empty clauses in $\bigcup_{k\geq 1} \mathcal{C}_k$ such that for every negative literal $\neg\exists G_2 \in \Gamma_2$ we have that $G_1 \not\models \neg\exists G_2$ and for every atomic literal $\forall(c : X \rightarrow G_2) \in \Gamma_2$ there is a monomorphism $h : X \rightarrow G_1$. Then $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}} \exists G[Q] \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G} = \{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \Gamma_2)\}$.*

Proof. As in the case of Proposition 8, we prove by induction that for all clauses Γ_1 and Γ_2, if $\exists G_1[Q_1] \vee \Gamma_1$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$ and for every literal $\neg\exists G_2 \in \Gamma_2$ we have that $G_1 \not\models \neg\exists G_2$ and for every literal $\forall(c : X \rightarrow G_2) \in \Gamma_2$ there is a monomorphism $h : X \rightarrow G_1$, then for every clause Γ_3 such that $\Gamma_2 \vee \Gamma_3$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$ we have that either Γ_2 is empty or $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}} \exists G[Q] \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G} = \{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \Gamma_2)\}$. This implies the proposition when Γ_3 is the empty clause.

If Γ_2 is the empty clause, then the proof is trivial. Otherwise, let us suppose that $\Gamma_2 = L \vee \Gamma_2'$, for a given literal L. By induction, we may assume that for any Γ_3 if $(\Gamma_2' \vee L \vee \Gamma_3) \in \bigcup_{k\geq 1} \mathcal{C}_k$ then either (a) Γ_2' is empty or (b) $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}'} \exists G[Q] \vee L \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}' = \{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \Gamma_2')\}$. Let us define Γ_3' to be equal to Γ_3 in case (a), and equal to $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}'} \exists G[Q] \vee \Gamma_3 \vee \Gamma_1)$ in case (b). Below we just consider the case where $L = \neg\forall(c : X \rightarrow C)$, since the proofs for the remaining cases are essentially identical to the proofs of the corresponding cases in Proposition 8.

- If $L = \neg\forall(c : X \rightarrow C)$ then we can apply the rule (R4') to $\exists G_1[Q_1] \vee \Gamma_1$ and $\neg\forall(c : X \rightarrow C) \vee \Gamma_3'$ yielding the clause $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}''} \exists G[Q] \vee \Gamma_1 \vee \Gamma_3') \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G}'' = \{\langle G, f_1\langle Q_1\rangle \cup \{\langle \neg\forall(g : X \rightarrow C_2), f_2\rangle\}\rangle \mid \langle f_1 : G_1 \rightarrow G \leftarrow X : f_2\rangle \in (G_1 \otimes X)\}$. Therefore, in case (a) we know that $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}''} \exists G[Q] \vee \Gamma_1 \vee \Gamma_3) \in \bigcup_{k\geq 1} \mathcal{C}_k$, and this completes the proof, since in this case, by definition, $\{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \Gamma_2)\} = \{\langle G, f_1\langle Q_1\rangle \cup \{\langle \neg\forall(g : X \rightarrow C_2), f_2\rangle\}\rangle \mid \langle f_1 : G_1 \rightarrow G \leftarrow X : f_2\rangle \in (G_1 \otimes X)\}$. On the other hand, in case (b) we know that $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}''} \exists G[Q] \vee \Gamma_1 \vee \bigvee_{\langle G,Q\rangle \in \mathcal{G}'} \exists G[Q] \vee \Gamma_3 \vee \Gamma_1) \equiv (\bigvee_{\langle G,Q\rangle \in \mathcal{G}''} \exists G[Q] \vee \bigvee_{\langle G,Q\rangle \in \mathcal{G}'} \exists G[Q] \vee \Gamma_3 \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$ and this completes the proof, since in this case, by definition, $\{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \Gamma_2)\} = \mathcal{G}'' \cup \mathcal{G}'$.

□

The proof of the proposition below is essentially identical to the proof of Proposition 9. For this reason, we will omit it.

Proposition 15 *Let $\mathcal{C}$ be a set of clauses , let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1'), (R2'), (R3'), (R4'), and (R5), and let $\exists G_1[Q_1] \vee \Gamma_1$ be a clause in $\bigcup_{k\geq 1} \mathcal{C}_k$, then for any $\mathcal{C}' \subseteq \mathcal{C}$, $(\bigvee_{\langle G,Q\rangle \in \mathcal{G}} \exists G[Q] \vee \Gamma_1) \in \bigcup_{k\geq 1} \mathcal{C}_k$, where $\mathcal{G} = \{\mathcal{G} = \{\langle G, Q\rangle \mid \langle h : G_1 \rightarrow G, Q\rangle \in I(\exists G_1[Q_1], \mathcal{C}')\}$.*

The precedence relation that we use here is basically the same one as the relation defined in the previous section. There are two main differences. The first one is that now the relation is defined on contextual literals. The second one is that now if $\exists G_1[Q_1] \prec \exists G_2[Q_1]$ then the context Q_1, when translated through $h_{G_1 \prec G_2}$, should be included in Q_2

Definition 16 *For every pair of contextual literals $\exists G_1[Q_1], \exists G_2[Q_2]$, $\exists G_1[Q_1] \prec \exists G_2[Q_2]$ if there is a monomorphism $h_{G_1 \prec G_2} : G_1 \rightarrow G_2$ and $h_{G_1 \prec G_2}\langle Q_1\rangle \subseteq Q_2$.*

Given a contextual literal $\exists C_1[Q_1]$, this literal precedes the results of $I(\exists C_1[Q_1], \mathcal{C})$ for any set of clauses $\mathcal{C}$:

Proposition 16 *If $\langle g : C_1 \rightarrow C2, Q_2\rangle$ is in $I(\exists C_1[Q_1], \mathcal{C})$ then $\exists C_1[Q_1] \prec \exists C_2[Q_2]$*

Proof. It is enough to define $h_{C_1 \prec C_2} = g$, since we know that, by Prop 11, $g\langle Q_1\rangle \subseteq Q_2$. □

As said above, we use this precedence relation to build (or to find) models of the given specification, like in the previous completeness proof. In particular, the notion of a *saturated* sequence is also a key concept. There are two main differences of the notion of saturated sequence needed here and the notion presented in the previous section. The first one is that here we also have to take into account the inferences with negative non-basic constraints. The second difference concerns the notion of closed literal. Intuitively, a closed literal is a literal that cannot lead to the

construction of a model of the given set of clauses. In the previous section, a literal was closed when it would not satisfy a strictly negative clause. In the current context a literal is considered also closed if it is inconsistent.

Definition 17 *Let $\mathcal{C}$ be a set of clauses, let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on on the rules (R1'), (R2'), (R3'), (R4'), and (R5), and let ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ be the set of all contextual literals occurring in clauses inferred in the refutation procedure. A literal $\exists G[Q]$ in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ is* closed *if either there is a strictly negative clause Γ in $\bigcup_{k\geq 1} \mathcal{C}_k$ such that $G \not\models \Gamma$ or if $\exists G[Q]$ is inconsistent. We also say that $\exists G[Q]$ is* open *if it is not closed.*

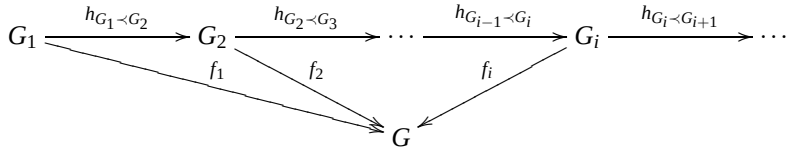
Then, the new definition of a saturated sequence, following the intuitions above is:

Definition 18 *An ascending sequence in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ $\exists G_1[Q_1] \prec \cdots \prec \exists G_i[Q_i] \prec \ldots$ is* saturated *if one of the following cases applies:*

- *the sequence is finite and its last element $\exists G_k[Q_k]$ satisfies that G_k is a model for $\mathcal{C}$, or*
- *the sequence is finite and its last element is closed, or*
- *the sequence is infinite, it consists only of open elements and for every clause Γ in $\bigcup_{k\geq 1} \mathcal{C}_k$ there is a literal L in Γ such that:*
  - (a) *if $L = \neg\exists C$, then for every j there is no monomorphism $m : C \to G_j$*
  - (b) *if $L = \exists C[Q]$, there is a j, such that there is a monomorphism $m : C \to G_j$*
  - (c) *If $L = \forall(c : X \to C)$ then for every i and every monomorphism $m : X \to G_i$ there is a j, with $i < j$, and a monomorphism $h : C \to G_j$ with $h_{C_i \prec C_j} \circ m = h \circ c$.*
  - (d) *if $L = \neg\forall(c : X \to C)$, then there is a j, such that $\langle \neg\forall(c : X \to C), h : X \to G_j \rangle$ is in Q_j for some monomorphism h.*

The lemma that shows that the colimit of infinite saturated sequences is a model of the given set of constraints is, again, very similar to the corresponding lemma in Section 3. However, the proof of the lemma below is slightly different to the prove of Lemma 8. In particular, here we have to consider the additional case of negative non-basic constraints. For this reason, below we include the proof for this case.

Lemma 8 *Let $\exists G_1[Q_1] \prec \cdots \prec \exists G_i[Q_i] \prec \ldots$ be an infinite saturated sequence in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ for a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ and let G be the colimit of the sequence:*



then G is a model for the given set of clauses, i.e. $G \models \mathcal{C}$.

Proof. Let Γ be any clause in $\mathcal{C}$. We have to prove that $G \models \Gamma$. Since the sequence is assumed to be saturated there should be a literal L in Γ such that the conditions (a), (b), (c) or (d) in Def. 18 are satisfied. The proof for the cases (a), (b), (c) is essentially identical to the corresponding proof in Lemma 8. For this reason we only include case (d):

(d) if $L = \neg\forall(c : X \to C)$, we know that there is a j, such that $\langle \neg\forall(c : X \to C), h : X \to G_j \rangle$ is in Q_j for some monomorphism h. As a consequence, $f_j \circ h$ is a monomorphism from X to G. Now we will prove that if there is a monomorphism $f : C \to G$, such that $f_j \circ h = f \circ c$, then some contextual literal $\exists G_k[Q_k]$ would be inconsistent, against the assumption that all the literals in the sequence are open (and therefore consistent). Let us suppose that such an f exists. Then, according to proposition 4 there must exist a monomorphism $f' : C \to G_i$ such that $f = f_i \circ f'$. Therefore, we would have $f_j \circ h = f \circ c = f_i \circ f' \circ c$. Now, we consider two cases. If $j \leq i$ then, we have that $f_j = f_i \circ h_{G_j \prec G_i}$ which means that $f_i \circ h_{G_j \prec G_i} \circ h = f_i \circ f' \circ c$. But since f_i is a monomorphism we have that $f' \circ c = h_{G_j \prec G_i} \circ h$. Now, according to the definition of the precedence relation:

$$\langle \neg\forall(c : X \to C), h_{G_j \prec G_i} \circ h : X \to G_i \rangle \in Q_i.$$

and this implies that $G_i[Q_i]$ would be inconsistent. If $i < j$ then $f_i = f_j \circ h_{G_i \prec G_j}$, which means that $f_j \circ h = f_i \circ$

$f' \circ c = f_j \circ h_{G_i \prec G_j} \circ f' \circ c$ and, again, since f_j is a monomorphism we have that $h = h_{G_i \prec G_j} \circ f' \circ c$, implying that $G_j[Q_j]$ would be inconsistent.

□

The proof of the lemma that shows us a procedure to define saturated infinite sequences is also a small variation of the proof of Lemma 3, where the only difference refers again to the case where negative non-basic constraints are considered.

Lemma 9 *Let $\exists G_1[Q_1] \prec \cdots \prec \exists G_i[Q_i] \prec \ldots$ be an infinite ascending sequence in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ for a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ such that, for every j, $\exists G_j[Q_j]$ is open and moreover $h_{G_j \prec G_{j+1}} \in I(\exists G_j[Q_j], \mathcal{C}_j)$. Then, $\exists G_1[Q_1] \prec \cdots \prec \exists G_i[Q_i] \prec \ldots$ is a saturated sequence.*

Proof. Let Γ be any clause in $\bigcup_{k\geq 1} \mathcal{C}_k$. More precisely, let us assume that $\Gamma \in \mathcal{C}_n$. We have to prove that there is a literal L in Γ such that:

(a) if $L = \neg\exists C$, then for every j there is no monomorphism $m : C \to G_j$
(b) if $L = \exists C[Q]$, there is a j, such that there is a monomorphism $m : C \to G_j$
(c) If $L = \forall(c : X \to C)$ then for every i and every monomorphism $m : X \to G_i$ there is a j, with $i < j$, and a monomorphism $f : C \to G_j$ with $h_{G_i \prec G_j} \circ m = f \circ c$.
(d) if $L = \neg\forall(c : X \to C)$, then there is a j, such that $\langle \neg\forall(c : X \to C), h : X \to G_j \rangle$ is in Q_j for some monomorphism h.

Now, if there is an L in Γ that satisfies condition (a), then the proposition trivially holds. Otherwise, assume that there is a j such that for every negative literal $L = \neg\exists C$ in Γ, there is a monomorphism $m : C \to G_j$. Let $k = max(n, j)$. By Proposition 13, we know that for every $k' \geq k$ there should be a literal L in Γ such that:

- if $L = \exists C[Q]$, then there is a monomorphism $m : C \to G_{k'+1}$.
- If $L = \forall(c : X \to C)$ then the proof is identical to the proof of the corresponding case in Lemma 3.
- if $L = \neg\forall(c : X \to C)$, then $\langle \neg\forall(c : X \to C), h : X \to G_{k'+1} \rangle \in Q_{k'+1}$.

□

The proof of the lemma for showing the existence of saturated sequences is also identical to the proof of Lemma 4.

Lemma 10 *Given a fair refutation procedure $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ if ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ is not empty then there is a saturated sequence in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$.*

The lemma that shows that if all saturated sequences are finite and end in a closed element then we can derive a clause consisting only of closed elements is now slightly different than Lemma 5.

Lemma 11 *Let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1), (R2), and (R3) such that $\mathcal{C}$ includes a clause Γ consisting only of basic positive literals and negative non basic literals. If every saturated sequence in ContLit$(\bigcup_{k\geq 1} \mathcal{C}_k)$ is finite and its last element is a closed literal then there is a clause Γ' in $\bigcup_{k\geq 1} \mathcal{C}_k$ consisting only of closed literals.*

Proof. First we will prove that if the clause Γ includes only basic positive literals and negative non basic literals, then there is a clause Γ' in $\bigcup_{k\geq 1} \mathcal{C}_k$ such that Γ' consists only of basic (contextual) literals.

Suppose that $\Gamma = \Gamma_1 \vee \Gamma_2$ where Γ_1 consists only of negative non basic constraints and Γ_2 consists only of contextual literals. We will prove our claim by induction on Γ_1. More precisely, we will prove that if Γ_1 consists only of negative non basic constraints then for every clause Γ_2 such that $\Gamma_1 \vee \Gamma_2$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$ there exists a clause Γ'_1 consisting only of contextual literals such that $\Gamma'_1 \vee \Gamma_2$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$:

- If Γ_1 is the empty clause then the case is trivial.
- If $\Gamma_1 = \neg\forall(c : X \to C) \vee \Gamma_3$ then, by induction, we know that there is a clause Γ'_3 consisting only of contextual literals such that $\neg\forall(c : X \to C) \vee \Gamma'_3 \vee \Gamma_2$ is in $\bigcup_{k\geq 1} \mathcal{C}_k$. Since we assume that every set of clauses includes the trivial true clause (i.e. the clause consisting only of the literal $\exists\emptyset$, which is equivalent to the contextual literal $\exists\emptyset[\emptyset]$), then we can apply rule (R4') to this trivial clause and to $\neg\forall(c : X \to C) \vee \Gamma'_3 \vee \Gamma_2$ inferring the clause: $\exists X[\{\langle \neg\forall(c : X \to C), id_X \rangle\}] \vee \Gamma'_3 \vee \Gamma_2$.

So we have shown that from every clause Γ including only contextual literals and negative non basic literals we can infer a clause Γ' including only contextual literals. It remains to show that from Γ' we can infer a clause including only closed literals. However, this proof is essentially identical to the proof of Lemma 5. □

We can finally show the completeness of our calculus. The proof follows, with small variations the proof of Lemma 6.

Lemma 12 ***(Completeness)*** *Let $\mathcal{C}$ be a set of atomic constraints, let $\mathcal{C}_0 \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over $\mathcal{C}$ based on the rules (R1'), (R2'), (R3'), (R4'), and (R5). If $\mathcal{C}$ is unsatisfiable then there is a j such that the empty clause is in $\mathcal{C}_j$.*

Proof. Suppose that the empty clause is not in $\mathcal{C}_j$ for any j. We have to show the existence of a graph G such that $G \models \mathcal{C}$. We consider four cases:

1. All clauses in $\mathcal{C}$ include some negative basic constraint, $\neg\exists C$, or some positive non basic constraint, $\forall(c : X \rightarrow C)$ (i.e., there is no clause Γ in $\mathcal{C}$ consisting only of basic positive literals and negative non-basic literals). In this case, the empty graph would satisfy all these positive atomic literals and all the negative basic literals and, as a consequence, would be a model for $\mathcal{C}$.
2. Otherwise, we have a clause Γ in $\mathcal{C}$ consisting only of basic positive literals and negative non-basic literals. Then, by Lemma 10, we know that there exist at least one saturated sequence in $\mathit{ContLit}(\bigcup_{k\geq 1}\mathcal{C}_k)$. By Def. 18, we have the following cases:
   (a) Every saturated sequence in $\mathit{ContLit}(\bigcup_{k\geq 1}\mathcal{C}_k)$ is finite and its last element is a closed literal. Using the same reasoning as in the proof of Lemma 6, we may see that this case is not possible.
   (b) There is a finite saturated sequence in $\mathit{ContLit}(\bigcup_{k\geq 1}\mathcal{C}_k)$ whose last element is a model for $\mathcal{C}$. The case is trivial.
   (c) There is an infinite saturated sequence $\exists G_1[Q_1] \prec \cdots \prec \exists G_i[Q_i] \prec \ldots$ in $\mathit{ContLit}(\bigcup_{k\geq 1}\mathcal{C}_k)$. Then, according to Lemma 8, its colimit is a model for the given set of clauses.

□

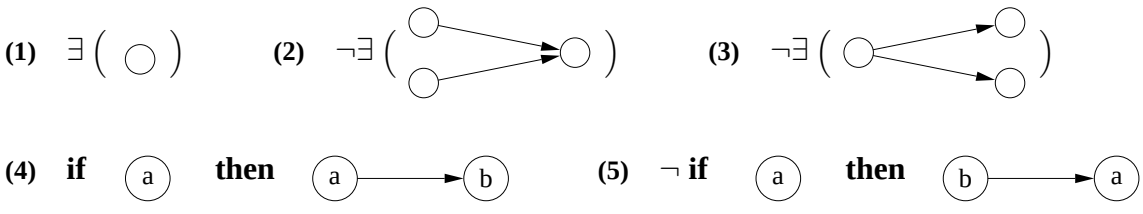
As a consequence of Lemmas 7 and 12, we have:

Theorem 2 ***(Soundness and Completeness)*** *Let $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ be a fair refutation procedure defined over a set of atomic constraints $\mathcal{C}$, based on the rules (R1'), (R2'), (R3'), (R4'), and (R5). Then, $\mathcal{C}$ is unsatisfiable if and only if there is a j such that the empty clause is in $\mathcal{C}_j$.*

As discussed above, our completeness results show that a set of constraints is satisfiable then a fair refutation procedure will never infer an empty clause from the given set of constraints. However, in the proof of completeness, the model constructed to show the satisfiability of the constraints may be an infinite graph. One could wonder whether in this situation it would always be possible to find an alternative finite model for these constraints. The answer is no. As we can see in the counter-example below, there are sets of atomic constraints which do not have finite models.

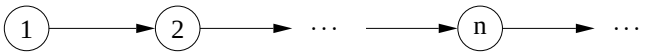
Theorem 3 ***(Finite satisfiability)*** *There are satisfiable sets of atomic constraints $\mathcal{C}$ such that there are no finite graphs G with $G \models \mathcal{C}$.*

Proof. The set of constraints below is not satisfied by any finite graph, but only by infinite graphs:



Let n be the number of nodes of a finite graph satisfying the constraints and e its number of edges. The first constraint specifies that the graph must have at least a node, i.e. $n \geq 1$. The second and third constraints specify that every node must have at most one incoming edge and one outgoing edge, i.e. $n \geq e$. The previous two constraints together with the fifth constraint (not every node has an incoming edge) imply that $n > e$. However, the fourth constraint

(every node has an outgoing edge) implies that $n \leq e$. Obviously no finite graph would satisfy these constraints. However the graph below does satisfy them:



□

5. Clause subsumption and elimination

Using the kind of refutation procedures that we have described, proving the unsatisfiability of a set of clauses $\mathcal{C}$ can be very costly. A main (standard) problem is that proving unsatisfiability implies doing an exhaustive search, considering all possible inferences among all the clauses. To reduce the cost of this search there are several possible approaches. For instance, one approach that we do not consider in this paper is to use some kind of ad-hoc heuristics or strategy to guide the search. A more general kind of solution is based on the elimination from the given specification of clauses or literals that may be known to be unnecessary for finding a refutation. In this way, we obviously reduce the search space. A technique that is often used for this purpose is *subsumption*. Intuitively, a clause Γ_1 subsumes Γ_2 if every refutation using Γ_2 can be replaced by a refutation using Γ_1. In this case, Γ_2 may be considered useless and we can delete it. The standard definition of clause subsumption applies also here, i.e. Γ_1 subsumes Γ_2 if every literal in Γ_1 subsumes a literal in Γ_2. However, the notion of literal subsumption is quite different from the standard notion of literal subsumption in first-order logic. In that case, a literal L_1 subsumes L_2 if there is a substitution σ that applied to the variables of L_1 yields L_2. This means that, in a sense, L_1 is smaller than L_2. Here, literal subsumption works exactly in the opposite direction. A literal $\exists C_1$ subsumes $\exists C_2$ if C_1 includes C_2:

Definition 19 ***(Literal and clause subsumption)*** *Given literals L_1 and L_2, we say that L_1 subsumes L_2, denoted $L_1 \lhd L_2$ if $L_1 = L_2$ or one of the following cases applies:*

- *If L_1 and L_2 are contextual constraints, $L_1 = \exists C_1[Q_1]$ and $L_2 = \exists C_2[Q_2]$, $f : C_2 \to C_1$ is a monomorphism, then $L_1 \lhd_f L_2$ if for every $\langle \neg\forall(c : X \to C), f_2 : X \to C_2\rangle$ in Q_2 there exists $\langle \neg\forall(c : X \to C), f_1 : X \to C_1\rangle$ in Q_1 such that $f_1 = f \circ f_2$. Moreover, $L_1 \lhd L_2$ if there exists an $f : C_2 \to C_1$ such that $L_1 \lhd_f L_2$.*
- *If L_1 and L_2 are basic negative constraints, $L_1 = \neg\exists C_1$ and $L_2 = \neg\exists C_2$, then $L_1 \lhd L_2$ if there is a monomorphism $h : C_1 \to C_2$.*
- *if L_1 and L_2 are positive atomic constraints, $L_1 = \forall(c_1 : X_1 \to C_1)$ and $L_2 = \forall(c2 : X_2 \to C_2)$, then $L_1 \lhd L_2$ if there are monomorphisms $g : X_1 \to X_2$ and $h : C_2 \to C_1$ such that $c_1 = h \circ c_2 \circ g$.*

Given clauses Γ_1 and Γ_2, $\Gamma_1 \lhd \Gamma_2$ if for every literal L_1 in Γ_1 there is a literal L_2 in Γ_2 such that $L_1 \lhd L_2$.

It may be noticed that we have not provided any explicit definition of subsumption for negative non-basic literals. This means that, implicitly, in this case subsumption coincides with equality. There are two reasons for this. On one hand, the most obvious candidate for this definition, the contravariant version of subsumption for positive non-basic literals does not work (i.e. Theorem 4 would not hold for that notion of subsumption). On the other hand, one of the main reasons for introducing subsumption, as explained below, is to have the possibility of eliminating (some of) the premises of a deduction rule after applying that inference. However, in this case, subsumption of negative non-basic literals plays no specific role for this elimination.

The following properties are a straightforward consequence of the above definition.

Proposition 17 *Subsumption satisfies the following properties:*

1. *If $\Gamma_1 \lhd \Gamma_2$ and Γ_2 is empty then Γ_1 is also empty.*
2. *If $\Gamma_1 \lhd \Gamma_2$ and $L_1 \lhd L_2$ then $\Gamma_1 \cup \{L_1\} \lhd \Gamma_2 \cup \{L_2\}$.*
3. *If $\Gamma_1 \lhd \Gamma_2$ and $\Gamma_1' \lhd \Gamma_2'$ then $\Gamma_1 \cup \Gamma_1' \lhd \Gamma_2 \cup \Gamma_2'$.*
4. *Given literals $\neg\forall(c : X \to C)$, $\exists C_1[Q_1]$, and $\exists C_2[Q_2]$. If $\exists C_1[Q_1] \lhd_f \exists C_2[Q_2]$ and there are monomorphisms $f_1 : X \to C_1$, $f_2 : X \to C_2$, such that $f_1 = f \circ f_2$, then $\exists C_1[Q_1'] \lhd \exists C_2[Q_2']$, where $Q_1' = Q_1 \cup \{\langle \neg\forall(c : X \to C), f_1 : X \to C_1\rangle\}$ and $Q_2' = Q_2 \cup \{\langle \neg\forall(c : X \to C), f_2 : X \to C_2\rangle\}$*
5. *Given literals $\exists C_1[Q_1]$, $\exists C_2[Q_2]$ and graphs C_1' and C_2'. If $\exists C_1[Q_1] \lhd_f \exists C_2[Q_2]$ and there are monomorphisms $f_1 : C_1 \to C_1'$, $f_2 : C_2 \to C_2'$, and $f' : C_2' \to C_1'$ such that $f_1 \circ f = f' \circ f_2$, then $\exists C_1'[f_1\langle Q_1\rangle] \lhd_{f'} \exists C_2'[f_1\langle Q_1\rangle]$.*

6. *If* $\exists C_1[Q_1] \lhd \exists C_2[Q_2]$, $\exists C_1'[Q_1'] \lhd \exists C_2'[Q_2']$, $\Gamma_1 = (\bigvee_{G_1 \in \mathcal{G}_1} \exists G_1[f_1\langle Q_1\rangle])$, *and* $\Gamma_2 = (\bigvee_{G_2 \in \mathcal{G}_2} \exists G_2[f_2\langle Q_2\rangle])$ *where* $\mathcal{G}_1 = \{G_1 \mid \langle f_1 : C_1 \to G_1 \leftarrow C_1' : f_1'\rangle \in (C_1 \otimes C_1')\}$ *and* $\mathcal{G}_2 = \{G_2 \mid \langle f_2 : C_2 \to G_2 \leftarrow C_2' : f_2'\rangle \in (C_2 \otimes C_2')\}$, *then* $\Gamma_1 \lhd \Gamma_2$.

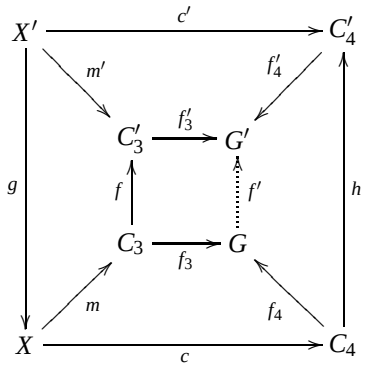
Now we can prove the main result of this section. Namely that subsumed clauses are not needed in refutations:

Theorem 4 *Let* Γ_1 *and* Γ_2 *be clauses, such that* $\Gamma_1 \lhd \Gamma_2$ *and let* $\mathcal{C} \Rightarrow \mathcal{C}_1 \Rightarrow \cdots \Rightarrow \mathcal{C}_k \ldots$ *be a fair refutation procedure where* $\Gamma_1, \Gamma_2 \in \mathcal{C}$. *There is a refutation* $\mathcal{C} \setminus \{\Gamma_2\} \Rightarrow \mathcal{C}_1' \Rightarrow \cdots \Rightarrow \mathcal{C}_k' \ldots$, *where* $\forall i\ \Gamma_2 \notin \mathcal{C}_i'$, *such that there is a k where the empty clause is in* $\mathcal{C}_k$ *if and only if there is a j where the empty clause is in* $\mathcal{C}_j'$

Proof. The if part is trivial since $\mathcal{C} \setminus \{\Gamma_2\}$ is included in $\mathcal{C}$, and this implies that any clause, including the empty clause, that can be inferred from $\mathcal{C} \setminus \{\Gamma_2\}$ can also be inferred from $\mathcal{C}$. To prove the only-if part, we build inductively the refutation $\mathcal{C} \setminus \{\Gamma_2\} \Rightarrow \mathcal{C}_1' \Rightarrow \cdots \Rightarrow \mathcal{C}_k' \ldots$ showing that for every k there is a j such that for every clause Γ in $\mathcal{C}_k$ there is a clause Γ' in $\mathcal{C}_j'$ such that $\Gamma' \lhd \Gamma$. This is enough to prove the theorem since, according to Prop. 17, the only clause that subsumes the empty clause is the empty clause, which means that if the empty clause is in $\mathcal{C}_k$ then the empty clause should also be in the $\mathcal{C}_j'$.

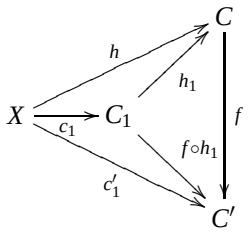
- The base case is trivial since the only clause that is in $\mathcal{C}$ which is not in $\mathcal{C} \setminus \{\Gamma_2\}$ is obviously Γ_2, and we have assumed that $\Gamma_1 \lhd \Gamma_2$ and $\Gamma_1 \in \mathcal{C}$.
- Let us assume that for every clause Γ_k in $\mathcal{C}_k$ there is a clause Γ_j' in $\mathcal{C}_j'$ such that $\Gamma_j' \lhd \Gamma_k$ and $j \leq k$. Now we consider five cases depending on the inference $\mathcal{C}_k \Rightarrow \mathcal{C}_{k+1} = \mathcal{C}_k \cup \{\Gamma\}$ in order to show that there is some Γ' in $\mathcal{C}_{j+1}'$ such that $\Gamma' \lhd \Gamma$:

  1. Γ is obtained applying rule (R1') to $\exists C_3[Q_3] \vee \Gamma_3$ and $\neg\exists C_4 \vee \Gamma_4$. This means that there exists a monomorphism $m : C_4 \to C_3$ and $\Gamma = \Gamma_3 \vee \Gamma_4$. By induction, we know that there should be clauses $\exists C_3'[Q_3'] \vee \Gamma_3'$ and $\neg\exists C_4' \vee \Gamma_4'$ in $\mathcal{C}_j'$ such that $\exists C_3'[Q_3'] \lhd \exists C_3[Q_3]$, $\Gamma_3' \lhd \Gamma_3$, $\neg\exists C_4' \lhd \neg\exists C_4$ and $\Gamma_4' \lhd \Gamma_4$. But this means that there are monomorphisms $h_4 : C_4' \to C_4$ and $h_3 : C_3 \to C_3'$. As a consequence $h_3 \circ m \circ h_4$ is a monomorphism from C_4' to C_3' which means that we can apply rule (R1') to the two clauses inferring the clause $\Gamma_3' \vee \Gamma_4'$ which, according to Proposition 17, subsumes $\Gamma_3 \vee \Gamma_4$. Therefore, we can define $\mathcal{C}_{j+1}' = \mathcal{C}_j' \cup \{\Gamma_3' \vee \Gamma_4'\}$.
  2. Γ is obtained applying rule (R2') to $\exists C_3[Q_3] \vee \Gamma_3$ and $\exists C_4[Q_4] \vee \Gamma_4$. Then $\Gamma = (\bigvee_{G \in \mathcal{G}} \exists G[f_3\langle Q_3\rangle \cup f_4\langle Q_4\rangle]) \vee \Gamma_3 \vee \Gamma_4$, where $\mathcal{G} = \{G \mid \langle f_3 : C_3 \to G \leftarrow C_4 : f_4\rangle \in (C_3 \otimes C_4)\}$. By induction, we know that there should be clauses $\exists C_3'[Q_3'] \vee \Gamma_3'$ and $\exists C_4'[Q_4'] \vee \Gamma_4'$ in $\mathcal{C}_j'$, such that $\exists C_3'[Q_3'] \lhd \exists C_3[Q_3]$, $\Gamma_3' \lhd \Gamma_3$, $\exists C_4'[Q_4'] \lhd \exists C_4[Q_4]$, and $\Gamma_4' \lhd \Gamma_4$. Then, we can apply the rule (R2') inferring the clause $\Gamma' = (\bigvee_{G \in \mathcal{G}} \exists G[f_3\langle Q_3'\rangle \cup f_4\langle Q_4'\rangle]) \vee \Gamma_3' \vee \Gamma_4'$, where $\mathcal{G} = \{G \mid \langle f_3 : C_3' \to G \leftarrow C_4' : f_4\rangle \in (C_3' \otimes C_4')\}$. Therefore, we can define $\mathcal{C}_{j+1}' = \mathcal{C}_j' \cup \{\Gamma'\}$, since according to Proposition 17, Γ' subsumes Γ.
  3. Γ is obtained applying rule (R3') to $\exists C_3[Q_3] \vee \Gamma_3$ and $\forall(c : X \to C_4) \vee \Gamma_4$. This means that there is a monomorphism $m : X \to C_3$ and $\Gamma = (\bigvee_{G \in \mathcal{G}} \exists G[f_3\langle Q_3\rangle]) \vee \Gamma_3 \vee \Gamma_4$, where $\mathcal{G}$ is the set consisting of all the graphs G such that there are two jointly surjective monomorphisms $f_3 : C_3 \to G$ and $f_4 : C_4 \to G$ such that $f_4 \circ c = f_3 \circ m$. By induction, we know that there should be clauses $\exists C_3'[Q_3'] \vee \Gamma_3'$ and $\forall(c' : X' \to C_4') \vee \Gamma_4'$ in $\mathcal{C}_j'$, such that $\exists C_3'[Q_3'] \lhd \exists C_3[Q_3]$, $\Gamma_3' \lhd \Gamma_3$, $\forall(c' : X' \to C_4') \lhd \forall(c : X \to C_4)$, and $\Gamma_4' \lhd \Gamma_4$. Moreover, this implies that there are monomorphisms $f : C_3 \to C_3'$, $g : X' \to X$ and $h : C_4 \to C_4'$ such that $c' = h \circ c \circ g$. But this means that there is a monomorphism $m' = f \circ m \circ g$ from X' to C_3'. As a consequence, we can apply the rule (R3') inferring the clause $\Gamma' = (\bigvee_{G' \in \mathcal{G}'} \exists G'[f_3'\langle Q_3'\rangle]) \vee \Gamma_3' \vee \Gamma_4'$, where $\mathcal{G}'$ is the set consisting of all the graphs G' such that there are two jointly surjective monomorphisms $f_3' : C_3' \to G'$ and $f_4' : C_4' \to G$ such that $f_4' \circ c' = f_3' \circ m'$. Now, for every $G' \in \mathcal{G}'$ we have monomorphisms $f_3' \circ f : C_3 \to G'$ and $f_4' \circ h : C_4 \to G'$ such that $f_3' \circ f \circ m = f_4' \circ h \circ c$. Then, by Prop. 2, there should be a $G \in \mathcal{G}$ and a monomorphism $f' : G \to G'$ such that the diagram below commutes.



But, according to Proposition 17, this means that for every $G' \in \mathcal{G}'$ there should be a $G \in \mathcal{G}$ such that $\exists G'[f'_3\langle Q'_3\rangle] \lhd \exists G[f_3\langle Q_3\rangle]$. Therefore, Γ' subsumes Γ.

4. Γ is obtained applying rule (R4') to $\exists C_3[Q_3] \vee \Gamma_3$ and $\neg\forall(g : X \to C) \vee \Gamma_4$. This means that $\Gamma = (\bigvee_{G\in\mathcal{G}} \exists G[Q]) \vee \Gamma_3 \vee \Gamma_4$, where $\mathcal{G} = \{G \mid \langle f_3 : C_3 \to G \leftarrow X : f_4\rangle \in (C_3 \otimes X)\}$, and $Q = f_3\langle Q_3\rangle \cup \{\langle\neg\forall(g : X \to C), f_4\rangle\}$. By induction, we know that there should be clauses $\exists C'_3[Q'_3] \vee \Gamma'_3$ and $\neg\forall(g : X \to C) \vee \Gamma'_4$ in $\mathcal{C}'_j$, such that $\exists C'_3[Q'_3] \lhd \exists C_3[Q_3]$, $\Gamma'_3 \lhd \Gamma_3$, and $\Gamma'_4 \lhd \Gamma_4$. As a consequence, we can apply the rule (R4') to these clauses inferring the clause $\Gamma' = (\bigvee_{G\in\mathcal{G}} \exists G[Q']) \vee \Gamma'_3 \vee \Gamma'_4$, where $\mathcal{G} = \{G \mid \langle f'_3 : C'_3 \to G \leftarrow X : f'_4\rangle \in (C'_3 \otimes X)\}$, and $Q' = f'_3\langle Q'_3\rangle \cup \{\langle\neg\forall(g : X \to C), f'_4\rangle\}$. Therefore, we can define $\mathcal{C}'_{j+1} = \mathcal{C}'_j \cup \{\Gamma'\}$, since according to Proposition 17, Γ' subsumes Γ.
5. Γ is obtained applying rule (R5) to $\exists C[Q] \vee \Gamma$. This means that $\exists C[Q]$ is inconsistent, i.e. there exists $\langle\neg\forall(c_1 : X \to C_1), h : X \to C\rangle$ in Q such that there is a monomorphism $h_1 : C_1 \to C$ such that $h = h_1 \circ c_1$. By induction, we know that there should be a clause $\exists C'[Q'] \vee \Gamma'$ in $\mathcal{C}'_j$, such that $\exists C'[Q'] \lhd \exists C[Q]$ and $\Gamma' \lhd \Gamma$. But this implies that there is a monomorphism $f : C \to C'$ and there is $\langle\neg\forall(c_1 : X \to C_1), h' : X \to C'\rangle$ in Q' such that $f \circ h = h'$, i.e.:



but this means that there is a monomorphism $f \circ h_1 : C_1 \to C'$. Moreover, we have that $h' = f \circ h = f \circ h_1 \circ c_1$ and, hence, $\exists C'[Q']$ is not consistent. Therefore, we can apply rule (R5) to the clause $\exists C'[Q'] \vee \Gamma'$ inferring Γ' and we can define $\mathcal{C}'_{j+1} = \mathcal{C}'_j \cup \{\Gamma'\}$.

□

It may be noted that, according to our definition of subsumption, in our inference rules (R1') - (R4') if one of the clauses Γ_1 or Γ_2 in the premises is empty then the result of the rule subsumes a premise. In the case of (R5) the result of the rule always subsumes the premise. For instance, given rule (R2'):

$$\frac{\exists C_1[Q_1] \vee \Gamma_1 \quad \exists C_2[Q_2] \vee \Gamma_2}{(\bigvee_{G\in\mathcal{G}} \exists G[f_1\langle Q_1\rangle \cup f_2\langle Q_2\rangle]) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R2')}$$

if Γ_1 is empty then the consequence of the rule subsumes $\exists C_2[Q_2] \vee \Gamma_2$. This means that after this inference, this premise could be eliminated since according to our previous theorem it is useless to find a refutation of our specifica-

tion. A similar thing happens with some of the inference rules that can be found below. If the corresponding Γ_1 or Γ_2 are empty then we can eliminate one of the premises.

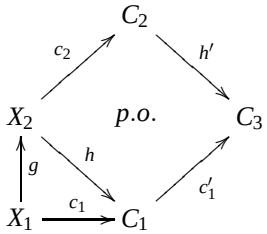
Example 5 *According to the previous theorem, we can delete all the subsumed clauses in a specification without losing completeness. In particular, in our running example, this means that the constraints (1) and (2) could be eliminated, since they are subsumed by clause (8). Clause (8) can also be eliminated since it is subsumed by clause (10). Clause (1) also subsumes clause (9), so we could also eliminate it. Finally, clause (10) can also be eliminated, since it is subsumed by clause (11).*

Another way of speeding up refutation procedures is to have inference rules, which perhaps are not needed for completeness, that allow us to infer clauses that may help us in finding shorter refutations, perhaps with the help of a heuristics or some proof-strategy. In this case, we may note that a positive (respectively, negative) literal which is *larger* (respectively, *smaller*) than another literal L_2 has better chances to be deleted from clauses. In addition, L_1 may subsume more literals than L_2 (and it may also subsume L_2). This means that if we are able to replace a clause including L_2 by a clause including L_1 we may have better chances of finding a refutation faster.

Below, we include several rules that may be used to find better clauses in the above sense. In particular for each of these clauses we prove its soundness. The first rule tells us that in a certain situation we may amalgamate two positive atomic constraints to create a new one that subsumes them.

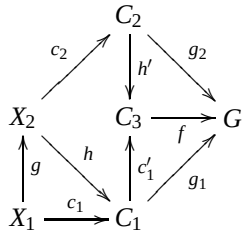
$$\frac{\forall(c_1 : X_1 \to C_1) \vee \Gamma_1 \quad \forall(c_2 : X_2 \to C_2) \vee \Gamma_2}{\forall(c_3 : X_1 \to C_3) \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R6)}$$

if there are monomorphisms $g : X_1 \to X_2$ and $h : X_2 \to C_1$ such that $c_1 = h \circ g$ and where C_3 and $c_3 = c_1' \circ c_1$ are defined by the pushout diagram below.



Proposition 18 *(R6) is sound.*

Proof. Suppose that $G \models \forall(c_1 : X_1 \to C_1) \vee \Gamma_1$, $G \models \forall(c_2 : X_2 \to C_2) \vee \Gamma_2$, and there exist monomorphisms $g : X_1 \to X_2$ and $h : X_2 \to C_1$ such that $c_1 = h \circ g$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Now, suppose $G \models \forall(c_1 : X_1 \to C_1)$ and $G \models \forall(c_2 : X_2 \to C_2)$ and suppose that there is a monomorphism $(f_1 : X_1 \to G)$ then we have to show that there should exist a monomorphism $(f : C_3 \to G)$ such that $f_1 = f \circ c_3$. Using that $G \models \forall(c_1 : X_1 \to C_1)$ we have that there is a monomorphism $(g_1 : C_1 \to G)$ with $f_1 = g_1 \circ c_1$. But this means that $g_1 \circ h$ is a monomorphism from X_2 to G using $G \models \forall(c_2 : X_2 \to C_2)$ we have that there is a monomorphism $(g_2 : C_2 \to G)$ such that $g_1 \circ h = g_2 \circ c_2$:



But, by the universal property of pushouts and Prop. 2, there must exist a monomorphism $(f : C_3 \to G)$ making the above diagram commute. But this means that $f_1 = g_1 \circ c_1 = f \circ c_1' \circ c_1 = f \circ c_3$. Hence, $G \models \forall(c_3 : X_1 \to C_3)$. □

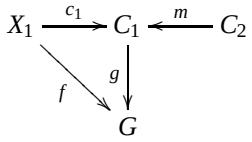
The last three rules describe the interaction of a negative and a positive constraint and, in this sense, they can be seen as generalizations of the rule (R1'). In particular, the rule (R7) describes the interaction of a negative basic constraint with a positive atomic constraint:

$$\frac{\forall(c : X_1 \to C_1) \vee \Gamma_1 \quad \neg\exists C_2 \vee \Gamma_2}{\neg\exists X_1 \vee \Gamma_1 \vee \Gamma_2} \quad \textbf{(R7)}$$

if there exists a monomorphism $m : C_2 \to C_1$

Proposition 19 *(R7) is sound.*

Proof. Suppose that $G \models \forall(c : X_1 \to C_1) \vee \Gamma_1$, $G \models \neg\exists C_2 \vee \Gamma_2$, and there exists a monomorphism $m : C_2 \to C_1$. The case where $G \models \Gamma_1$ or $G \models \Gamma_2$ is trivial. Now, suppose that $G \models \forall(c : X_1 \to C_1)$ and $G \models \neg\exists C_2$, and suppose that there exists a monomorphism $f : X_1 \to G$ this means that there should be a monomorphism $g : C_1 \to G$ with $g \circ c_1 = f$. But this means that $g \circ m$ is a monomorphism from C_2 to G:



Therefore such f cannot exist, which means that $G \models \neg\exists X_1$ □

Rule (R8) can be seen as a variation of rule (R1'). In particular, as we have discussed above a negative atomic constraint $\neg\forall(c : X_1 \to C_1)$ can be seen as a variation of the basic constraint $\exists X_1$ in the sense that, in both cases we are asking that the graph X_1 should be included in the given graph.

$$\frac{\neg\forall(c : X_1 \to C_1) \vee \Gamma_1 \quad \neg\exists C_2 \vee \Gamma_2}{\Gamma_1 \vee \Gamma_2} \quad \textbf{(R8)}$$

if there exists a monomorphism $m : C_2 \to X_1$

Proposition 20 *(R8) is sound.*

Proof. It is enough to see that a graph G cannot satisfy simultaneously $\neg\forall(c : X_1 \to C_1)$ and $\neg\exists C_2$. The reason is that if $G \models \neg\forall(c : X_1 \to C_1) \vee \Gamma_1$ this implies that there should exist a monomorphism $f : X_1 \to G$. But this would imply the existence of the monomorphism $f \circ m$ from C_2 to G. □

Finally, rule (R9) can be seen as a variation of resolution when considering atomic constraints:

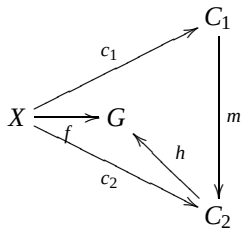
$$\frac{\neg\forall(c_1 : X \to C_1) \vee \Gamma_1 \quad \forall(c_2 : X \to C_2) \vee \Gamma_2}{\Gamma_1 \vee \Gamma_2} \quad \textbf{(R9)}$$

if there is a monomorphism $m : C_1 \to C_2$ such that $c_2 = m \circ c_1$.

Proposition 21 *(R9) is sound.*

Proof. It is enough to see that if there is a monomorphism $m : C_1 \to C_2$ such that $c_2 = m \circ c_1$ then a graph G cannot satisfy simultaneously $\neg\forall(c_1 : X \to C_1)$ and $\forall(c_2 : X \to C_2)$. The reason is that if $G \models \neg\forall(c_1 : X \to C_1)$ this implies

that there should exist a monomorphism $f : X \to G$ such that there is no $h_1 : C_1 \to G$ with $h_1 \circ c_1 = f$. But $G \models \forall(c_2 : X \to C_2)$ implies the existence of a monomorphism $h : C_2 \to G$, such that $f = h \circ c_2$:



But this would mean that we have the monomorphism $h \circ m$ from C_1 to G. Moreover, we know that $f = h \circ c_2 = h \circ m \circ c_1$ in contradiction with the side condition above. □

6. Conclusion

In this paper we have shown how we can use graph constraints as a specification formalism to define constraints associated to visual modelling formalisms or to specify classes of semi-structured documents. In particular, we have shown how we can reason about these specifications, providing inference rules that are sound and complete. Moreover, as can be seen in our examples, and as a consequence of Lemmas 3 and3, the completeness proofs show that our inference rules can also be used for the construction of models for the given sets of constraints.

Our results apply not only to plain graphs, but generalize to a large class of structures including typed and attributed graphs. In this sense, in [EEHP04, EEPT06] the constraints that we consider have been defined for any adhesive HLR-category [LaS04, EEPT06]. However, to be precise, to generalize our results, we would need that the underlying category of structures satisfies the properties stated in Section 2.1, which are used in the main results in the paper, and that are not considered in [EEHP04, EEPT06]. First, we would need that $G_1 \otimes G_2$ is finite, provided that G_1 and G_2 are finite. Second, we would need that our category satisfies the property of pair factorization as stated in proposition 1. Finally, we would need that the given category satisfies the existence of infinite colimits as stated in Prop. 4. In this sense, most set-based categories, in particular, most graph categories satisfy these conditions. However, the category of attributed graphs presents some problems. In particular, in general, if G_1 and G_2 are arbitrary attributed graphs then $G_1 \otimes G_2$ may be infinite, even if the graph part of G_1 and G_2 is finite. It is enough that the set of possible values for the attributes is infinite. However, if the definition of $G_1 \otimes G_2$ is restricted to the case where the jointly surjective morphisms are not only monomorphisms, but also the identity on the attributes, then $G_1 \otimes G_2$ would be finite. These monomorphisms are called M-morphisms in [EEPT06] and are needed to prove that attributed graphs are an adhesive HLR-category. Therefore, in this case it would be enough to show that the factorization properties hold for M-morphisms. Nevertheless, in [Ore08] we have studied constraints on attributed graphs following a completely different approach, which is inspired in the area of Constraint Logic Programming. The basic idea is to consider that an attributed graph (and therefore an attributed graph constraint) can be seen as a graph labelled with variables together with a logical formula on this variables. This allowed us to make a neat separation between the graph part and the data part of attributed graph constraints and to provide sound and complete inference rules which are quite close to the ones presented in this paper, but that may ask, as a side condition, for the satisfiability of some formulas of the data.

Further work is concerned, on one hand, to the extension of this results to the case of arbitrary nested constraints, and on the other, with the implementation of our techniques. In particular, we think that it will not be too difficult to implement them on top of the AGG system [Tae04], given that the basic construction that we use in our inference rules (i.e. building $G_1 \otimes G_2$) is already implemented there.

Acknowledgements This work has been partially supported by the CICYT project FORMALISM (ref. TIN2007-66523) and by the AGAUR grant to the research group ALBCOM (ref. 00516). Part of the work was done during a sabbatical leave of the first author at TU Berlin with financial support from the Ministerio de Ciencia e Innovación (grant ref. PR2008-0185).

References

[Tae04] Taentzer, G, AGG: A Graph Transformation Environment for Modeling and Validation of Software, in: J. Pfaltz, M. Nagl and B. Boehlen, editors, *Application of Graph Transformations with Industrial Relevance (AGTIVE03)*, LNCS 3062, Springer, 2004 pp. 446 456. URL: http//tfs.cs.tu-berlin.de/agg.

[AlF06] Alpuente M, Ballis D, and Falaschi M: Automated Verification of Web Sites Using Partial Rewriting. *Software Tools for Technology Transfer*, 8 (2006), 565–585.

[BCKL06] Baldan P, Corradini A, Koenig B, Lluch-Lafuente A: A Temporal Graph Logic for Verification of Graph Transformation Systems. In *Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006* . Springer Lecture Notes in Computer Science 4409 (2007), 1–20

[Cou97] Courcelle B: The expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic, in [Roz97] (1997), 313–400.

[EEPT06] Ehrig H, Ehrig K, Prange U, Taentzer G: *Fundamentals of Algebraic Graph Transformation*, Springer (2006).

[EEHP04] Ehrig E, Ehrig K, Habel A, Pennemann KH: Constraints and Application Conditions: From Graphs to High-Level Structures. In *Graph Transformations, Second International Conference, ICGT 2004* (Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, Grzegorz Rozenberg, Eds.), Springer Lecture Notes in Computer Science 3256 (2004), 287–303.

[EhH86] Ehrig H, Habel A: Graph Grammars with Application Conditions. In *The Book of L* (Grzegorz Rozenberg and Arto Salomaa, Eds.), Springer (1986), 87–100.

[EEFN03] Ellmer E, Emmerich W, Finkelstein A, and Nentwich C: Flexible Consistency Checking. *ACM Transaction on Software Engineering and Methodology*, 12(1) (2003), 28–63.

[HHT96] Habel A, Heckel R, Taentzer G: Graph Grammars with Negative Application Conditions. Fundam. Inform. 26(3/4): 287–313 (1996).

[HaP05] Habel A, Pennemann KH: Nested Constraints and Application Conditions for High-Level Structures. In *Formal Methods in Software and Systems Modeling, Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday* (Hans-Joerg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, Gabriele Taentzer, Eds.), Springer Lecture Notes in Computer Science 3393 (2005), 293–308.

[HaP06] Habel A, Pennemann KH: Satisfiability of High-Level Conditions. In *Graph Transformations, Third International Conference, ICGT 2006* (Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, Grzegorz Rozenberg, Eds.), Springer Lecture Notes in Computer Science 4178 (2006), 430–444.

[HaP08] Habel A, Pennemann KH: Correctness of high-level transformation systems relative to nested conditions. *Math. Struct. in Comp. Sc.* (2008). Accepted for publication.

[HeW95] Heckel R and Wagner A: Ensuring Consistency of Conditional Graph Grammars - A Constructive Approach -. In Proceedings SEGRAGRA 1995, *Electr. Notes Theor. Comput. Sci.*, Volume 2 (1995), 118–126.

[Jel00] Jelliffe R: “Schematron”, Internet Document, May 2000. (http://xml.ascc.net/resource/schematron/).

[LaS04] Lack S, Sobocinski P: Adhesive Categories. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004* (Igor Walukiewicz, Ed.), Springer Lecture Notes in Computer Science 2987 (2004), 273–288.

[LEO06] Lambers L, Ehrig H, Orejas F: Conflict Detection for Graph Transformation with Negative Application Conditions. In *Graph Transformations, Third International Conference, ICGT 2006*, (Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, Grzegorz Rozenberg, Eds.), Springer Lecture Notes in Computer Science 4178 (2006), 61–76.

[LaG08] De Lara J, Guerra E: Pattern-Based Model-to-Model Transformation. In *Graph Transformations, 4th International Conference, ICGT 2008* (Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, Gabriele TaentzerEds.), Springer Lecture Notes in Computer Science 5214 (2008), 426–441

[Llo87] Lloyd JW: *Foundations of Logic Programming (2nd edition)*. Springer-Verlag 1987.

[Ore08] Orejas F: Attributed Graph Constraints. In *Graph Transformations, 4th International Conference, ICGT 2008* (Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, Gabriele TaentzerEds.), Springer Lecture Notes in Computer Science 5214 (2008), 274–288.

[OEP08] Fernando Orejas, Hartmut Ehrig, Ulrike Prange: A Logic of Graph Constraints. In *Fundamental Approaches to Software Engineering, 11th International Conference, FASE 2008*, Jose Luiz Fiadeiro, Paola Inverardi (Eds.). Springer Lecture Notes in Computer Science 4961 (2008) 179-198

[KMP05] Koch M, Mancini LV, Parisi-Presicce F: Graph-based specification of access control policies. *J. Comput. Syst. Sci.* 71(1): 1-33 (2005)

[Pen08] Pennemann KH: Resolution-like Theorem Proving for High-Level Conditions. In *Graph Transformations, 4th International Conference, ICGT 2008* (Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, Gabriele TaentzerEds.), Springer Lecture Notes in Computer Science 5214 (2008), 289–304.

[Ren04] Rensink A: Representing First-Order Logic Using Graphs. In *Graph Transformations, Second International Conference, ICGT 2004* (Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, Grzegorz Rozenberg, Eds.), Springer Lecture Notes in Computer Science 3256 (2004), 319–335.

[Roz97] Rozenberg, G (ed.): *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 1 Foundations*, World Scientific, 1997.