A Strict Constrained Superposition Calculus for Graphs

Rachid Echahed, Mnacho Echenim, Mehdi Mhalla, and Nicolas Peltier(✉)

Université Grenoble Alpes, LIG, CNRS, Inria, Grenoble INP,
38000 Grenoble, France
nicolas.peltier@imag.fr

Abstract. We propose a superposition-based proof procedure to reason on equational first order formulas defined over graphs. First, we introduce the considered graphs that are directed labeled graphs with lists of roots standing for pins or interfaces for replacements. Then the syntax and semantics of the considered logic are defined. The formulas at hand are clause sets built on equations and disequations on graphs. Afterwards, a sound and complete proof procedure is provided, and redundancy criteria are introduced to dismiss useless clauses and improve the efficiency of the procedure. In a first step, a set of inferences rules is provided in the case of uninterpreted labels. In a second step, the proposed rules are lifted to take into account labels defined as terms interpreted in some arbitrary theory. Particular formulas of interest are Horn clauses, for which stronger redundancy criteria can be devised. Essential differences with the usual term superposition calculus are emphasized.

1 Introduction

Graphs are ubiquitous structures in computer science. They are used to model several notions such as data, program runs (transition systems), networks, software and hardware architectures. They are also often used as foundational structures to model knowledge or data bases, cognitive or intelligent systems as well as physical, chemical or biological phenomena. They constitute, in addition, the basis of operational research or combinatorics. Graphs are, definitely, fundamental structures for modelling, computing and reasoning. Graph transformations have been studied since the early 70's [29]. Some of their applications can be found in [16,18]. In the literature, one can distinguish two main streams of approaches for graph transformation, namely the algebraic approaches [15,12] where category theory is used to define structure transformations in a very abstract and elegant way and the algorithmic approaches where graph transformations are defined by means of the actual algorithms involved in the transformations [20,13].

During the last decade, a very interesting application of graph transformations has emerged in the area of quantum models of computation, see e.g., the calculi ZX [11], ZH [3], ZW [24] or PBS [10]. In these calculi, one can specify quantum algorithms using particular graphs and can make some equational reasoning on them to verify correctness of quantum algorithms, see e.g. the

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 135–155, 2023.
https://doi.org/10.1007/978-3-031-30829-1_7

Quantomatic tool [25]. In such situations, making automated equational reasoning over graphs is very desirable even though equational theories over graphs are not recursively enumerable in general (see e.g. [7]).

The *superposition calculus* [1] is one of the most successful automated proof procedures which handles equational theories (on terms) which is being actually implemented in various theorem provers such as Vampire [28], Spass [32], or E [30]. The calculus operates on finite sets of equational clauses. It is defined as a set of *inference rules*, which deduce new clauses from previous ones. To prune the search space, strong restrictions (based on term orderings and literal selection functions) are imposed on the inferences, and redundancy criteria are provided to detect and dismiss useless clauses. The rules are applied until a contradiction (i.e., the empty clause) is derived or until the set is *saturated*, i.e., no further non-redundant clause may be deduced. The calculus is *refutationally complete*, in the sense that it is able to derive a contradiction from any unsatisfiable clause set. In a recent work [14], we proposed a superposition calculus for testing the unsatisfiability of sets of equations and disequations between graphs whose shapes are inspired by those used in the ZX calculus, where nodes are labeled by first-order (uninterpreted) terms. In the present paper we extend this work in several directions: (i) We tackle full clauses, i.e., disjunctions of equations and disequations. This extension turned out to be much more difficult than we initially expected, due to the fact that no reduction order exists on the considered graphs (see Examples 19 and 22), which complicates the completeness proof. We introduce redundancy criteria that cover some usual deletion and simplification rules. (ii) We lift the obtained calculus into a constrained calculus operating on graphs labeled by terms interpreted in some base theory. The procedure is a semi-decision procedure for unsatisfiability if the underlying theory is (semi) decidable and compact. (iii) We consider a slightly different class of graphs, where multi-edges are allowed. The new framework has the advantage of being both more general and simpler, and it also improves the efficiency of the calculus (more precisely for the computation of "merges" between graphs, see Remark 9).

Why defining a graph superposition calculus is difficult. We wish to emphasize some important differences between term and graph superposition. (i) It is well-known that term rewrite systems that are terminating and in which all critical pairs are joinable are confluent. This property plays a key rôle in the completeness proof of the superposition calculus. However, such a property does *not* hold for graph rewrite systems, and, worse, confluence is undecidable for terminating graph rewrite rules (if confluence is meant modulo isomorphism). As it is done in [14] we overcome this issue by considering a special class of graphs, for which the above property holds. This class is obtained by restricting the way graphs can be composed and replaced, using a sequence of distinguished nodes in the graphs, called roots. (ii) The usual superposition calculus is based on the use of a *reduction order*, i.e., a well-founded order on terms that is total on ground terms and closed under instantiation and embedding. Unfortunately no such order exists for graphs in general (see Example 19). Thus the model construction algorithm used to establish refutational completeness must cope with non terminating systems (indeed, since a ground equation $\mathfrak{g} \approx \mathfrak{h}$ cannot always be oriented, one must consider both rules: $\mathfrak{g} \to \mathfrak{h}$ and $\mathfrak{h} \to \mathfrak{g}$, which entails that the system does not terminate). Confluence is harder to establish for non terminating systems and we need to devise a new confluence criterion. (iii) The usual redundancy criterion of [1] (where a clause is considered redundant if it is implied by smaller clauses) does not apply to graphs. For instance the conclusion of an inference may be strictly bigger than all the premises (see Example 21). This is due to the fact that two graphs may overlap without one of them being included in the other. Such a behavior cannot be avoided, since, as proven in [14, Theorem 45], satisfiability is undecidable for sets of ground equational clauses defined on graphs (whereas it is well known to be decidable for standard ground clauses based on terms), thus superposition cannot terminate on ground graphs. Furthermore, we show (see Example 22) that the calculus is – rather surprisingly – not compatible with tautology deletion in general (tautology deletion is possible for Horn clauses).

Related work. The graphs we are considering are intended to capture (possibly cyclic) circuit shaped structures such as those used in the ZX or related calculi. They are close to hypergraphs with interfaces as used in some papers (see, e.g. [5]) where the roots or interfaces are used in the gluing process while transforming a graph. We follow an algorithmic approach when transforming the graphs. This approach eases the completeness proofs of the proposed superposition calculus. However, the performed graph transformations used in the present paper can be encoded as simple double pushout (DPO) [19] steps of the form $L \longleftarrow \mathit{Roots} \longrightarrow R$ with some additional constraints on matched subgraphs. It is also a particular case of DPOI steps (DPO with interfaces) where the roots play the rôle of the interfaces [5]. Automated reasoning in presence of graph structures is not an easy task in general. Several authors did tackle this problem and one can distinguish different approaches in the literature. Variants of Hoare-like calculi have been proposed for the verification of graph transformation systems see, e.g., [23,26,6,8]. Likewise, model checking procedures have also been devised in presence of graph structures see, e.g. [27,31]. In these works, a dynamic logic underlying program execution is assumed. In addition, a dedicated logic is used to express graph properties to be proven. Other techniques have been used to prove graph equivalences such as bisimulation [17] or normalization using terminating and confuent graph rewriting systems [9]. In the paper at hand, we are rather concerned by a refutational proof technique based on superposition dedicated to a class of graphs. Thus our proof procedure departs from all the aforementioned works. To our knowledge, only the report [22] presents a refutational procedure dedicated to ZX diagrams which is close to ours. However, the authors use the classical superposition calculus [1] over first-order terms and provide a translation from the considered graphs to first-order terms. Such translation needs the use of additional axioms encoding some graph properties such as associativity and commutativity of graph constructor operations. Such additional axioms are useless in our framework. The class of graph rewriting systems handled in our proof procedure are not necessarily terminating and thus we had to devise new criteria to ensure their (ground) confluence instead of using joinability of pre-critical pairs as done in [4].

The paper is organized as follows. Section 2 introduces some basic notations and defines the considered graphs and the operations used over them. In Section 3 the syntax and semantics of the formulas are introduced. In Section 4, a first set of inference rules is defined to test the satisfiability of sets of clauses where graphs are endowed with uninterpreted labels and its completeness is established modulo a redundancy criterion that captures usual deletion or simplification rules (such as subsumption). In Section 5 the obtained calculus is lifted to graphs labeled with terms that can be interpreted in some arbitrary theory and possibly containing variables. Completeness is guaranteed if the theory is semi-decidable and compact. This last calculus is proven complete and an enhanced redundancy test is proposed. Concluding remarks are given in Section 6. Due to lack of space, proofs are omitted.

2 Graphs and Graph Operations

We briefly review some usual definitions and notations. For any partial function f, we denote by $dom(f)$ the domain of f. If f and g are partial functions, we write $f(x) = g(x)$ to state that either $x \notin dom(f) \cup dom(g)$ or that $x \in dom(f) \cap dom(g)$ and the images of x by f and g are identical. Given a multiset $\mathfrak{m}$ and an element e, $\mathfrak{m}(e)$ denotes the multiplicity of e in $\mathfrak{m}$. For all multisets $\mathfrak{m}_1$ and $\mathfrak{m}_2$, we denote by $\mathfrak{m}_1 + \mathfrak{m}_2$ and $\mathfrak{m}_1 - \mathfrak{m}_2$ the sum and difference of $\mathfrak{m}_1$ and $\mathfrak{m}_2$, respectively. We write $\mathfrak{m}_1 \sqsubseteq \mathfrak{m}_2$ to state that $\mathfrak{m}_1$ is included in $\mathfrak{m}_2$. A multiset containing exactly the elements $e_1, \dots, e_n$ is written $\{e_1, \dots, e_n\}$. We denote by $\mathfrak{m}_1 \sqcup \mathfrak{m}_2$ the union of $\mathfrak{m}_1$ and $\mathfrak{m}_2$ (i.e., the minimal multiset containing $\mathfrak{m}_1$ and $\mathfrak{m}_2$) defined as follows: for all elements e, $(\mathfrak{m}_1 \sqcup \mathfrak{m}_2)(e) = \max(\mathfrak{m}_1(e), \mathfrak{m}_2(e))$. Finite sequences may sometimes be identified with sets if the order is not important, e.g., if $\boldsymbol{y} = (y_1, \dots, y_n)$, we may write $x \in \boldsymbol{y}$ to state that $x = y_i$, for some $i = 1, \dots, n$. We recall that a preorder is a binary relation that is reflexive and transitive. Any preorder $\leq$ may be associated with a strict order $<$ defined as follows: $x < y \iff (x \leq y \wedge y \not\leq x)$.

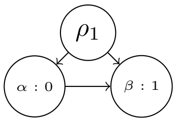
The graphs we consider are directed, labeled graphs enriched with a sequence of distinguished nodes, called *roots*:

Definition 1. *Let $\mathcal{N}$ be a countably infinite set of* nodes *and let $\mathcal{L}$ be a set of* labels*, disjoint from $\mathcal{N}$. An $\mathcal{L}$-*graph *$\mathfrak{g}$ is a tuple $\langle N, E, R, L\rangle$, where:*

- *$N \subseteq \mathcal{N}$ is a finite set of nodes in $\mathcal{N}$, called* vertices *or* nodes*;*
- *E is a finite multiset of pairs in $N \times N$, called* edges*;*
- *R is a sequence of nodes in N, with no repetition, called the* roots *of $\mathfrak{g}$;*
- *L is a function mapping every node in $N \setminus R$ to a* label *in $\mathcal{L}$.*

The components N, E, R and L of a graph $\mathfrak{g}$ are denoted by $N_\mathfrak{g}$, $E_\mathfrak{g}$, $R_\mathfrak{g}$ and $L_\mathfrak{g}$, respectively. We denote by $\widehat{N}_\mathfrak{g}$ the set of nodes $\alpha \in N_\mathfrak{g}$ that do not occur in $R_\mathfrak{g}$. The profile *of a graph $\mathfrak{g}$, written $pr(\mathfrak{g})$, is the length of $R_\mathfrak{g}$.*

Example 2. The $\mathcal{L}$-graph $\mathfrak{g}$ with $N_\mathfrak{g} = \{\rho_1, \alpha, \beta\}$, $E_\mathfrak{g} = \{(\rho_1, \alpha), (\rho_1, \beta), (\alpha, \beta)\}$, $R_\mathfrak{g} = (\rho_1)$, $dom(L_\mathfrak{g}) = \{\alpha, \beta\}$, $L_\mathfrak{g}(\alpha) = 0$ and $L_\mathfrak{g}(\beta) = 1$ is depicted graphically as follows:



We write $\alpha : \ell$ to state that a node named α is labeled by ℓ. In many cases, the names of the non-root nodes will be irrelevant, and will thus be omitted. When possible, root nodes will be named $\rho_1, \rho_2, \rho_3, \ldots$ in this order.

In the following, $\mathcal{L}$-graphs will be considered up to a renaming of nodes. More precisely, the isomorphism relation on $\mathcal{L}$-graphs is defined as follows.

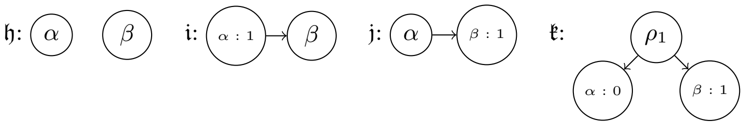
Definition 3. *An $\mathcal{N}$-*renaming *μ is an injective mapping from $\mathcal{N}$ to $\mathcal{N}$. It is extended to any $\mathcal{L}$-graph $\mathfrak{g}$ by replacing every occurrence of a node α by $\mu(\alpha)$. In particular, the function $L_{\mu(\mathfrak{g})}$ is defined as follows: $L_{\mu(\mathfrak{g})}(\alpha) = \ell$ iff $L_\mathfrak{g}(\beta) = \ell$ for some $\beta \in N_\mathfrak{g}$ such that $\mu(\beta) = \alpha$ ($L_{\mu(\mathfrak{g})}$ is well-defined since μ is injective). We write $\mathfrak{g} \equiv \mathfrak{h}$ if $\mathfrak{h} = \mu(\mathfrak{g})$, for some $\mathcal{N}$-renaming μ. It is easy to check that $\equiv$ is an equivalence relation. Two $\mathcal{L}$-graphs $\mathfrak{g}, \mathfrak{h}$ such that $\mathfrak{g} \equiv \mathfrak{h}$ are* isomorphic.

2.1 Subgraphs and Replacement

We define the notion of a subgraph. The definition is slightly stronger than the usual one in graph theory because it imposes that only nodes that are roots in the subgraph can be connected to a node outside the subgraph. These roots can be viewed as an "interface" which restricts the way graphs may be connected and composed.

Definition 4 (Subgraph). *A graph $\mathfrak{h}$ is a* subgraph *of $\mathfrak{g}$ (written $\mathfrak{h} \leq^g \mathfrak{g}$) if $N_\mathfrak{h} \subseteq N_\mathfrak{g}$, $E_\mathfrak{h} \sqsubseteq E_\mathfrak{g}$, $\widehat{N}_\mathfrak{h} \subseteq \widehat{N}_\mathfrak{g}$, $L_\mathfrak{h}(\alpha) = L_\mathfrak{g}(\alpha)$ for all $\alpha \in \widehat{N}_\mathfrak{h}$ and if a node α occurs in an edge in $E_\mathfrak{g} - E_\mathfrak{h}$ then $\alpha \notin \widehat{N}_\mathfrak{h}$.*

Example 5. Consider the $\mathcal{L}$-graphs $\mathfrak{h}$, $\mathfrak{i}$, $\mathfrak{j}$ and $\mathfrak{k}$ with respective roots (α, β), (β), (α) and (ρ_1), defined as follows:



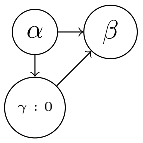
The $\mathcal{L}$-graph $\mathfrak{h}$ is a subgraph of the $\mathcal{L}$-graph $\mathfrak{g}$ from Example 2, but $\mathfrak{i}$, $\mathfrak{j}$ and $\mathfrak{k}$ are not. Indeed, α has different labels in $\mathfrak{g}$ and $\mathfrak{i}$; $\mathfrak{g}$ contains an edge between ρ_1 and β that does not occur in $\mathfrak{j}$ and β is not a root node in $\mathfrak{j}$; and $E_\mathfrak{g} - E_\mathfrak{k}$ contains the edge (α, β) between nodes that are not roots in $\mathfrak{k}$.

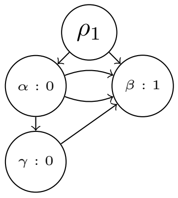
The replacement operation is defined in a natural way: all vertices and edges occurring from the replaced subgraph are deleted and replaced by those in the replacing graph (we assume that the considered graphs share the same roots).

Definition 6 (Subgraph replacement). *Let $\mathfrak{g}$ be an $\mathcal{L}$-graph and let $\mathfrak{h}$ be a subgraph of $\mathfrak{g}$. An $\mathcal{L}$-graph $\mathfrak{i}$ is* substitutable for $\mathfrak{h}$ in $\mathfrak{g}$ *if $R_\mathfrak{i} = R_\mathfrak{h}$ and $N_\mathfrak{g} \cap \widehat{N}_\mathfrak{i} = \emptyset$. If $\mathfrak{i}$ is substitutable for $\mathfrak{h}$ in $\mathfrak{g}$, then we denote by $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}\}$ (the $\mathcal{L}$-graph obtained by replacing $\mathfrak{h}$ by $\mathfrak{i}$ in $\mathfrak{g}$) the tuple $\langle N', E', R', L' \rangle$, where:*

- $N' \stackrel{def}{=} (N_\mathfrak{g} \setminus N_\mathfrak{h}) \cup N_\mathfrak{i}$. *Note that since $R_\mathfrak{i} = R_\mathfrak{h}$ we have $N' = (N_\mathfrak{g} \setminus \widehat{N}_\mathfrak{h}) \cup \widehat{N}_\mathfrak{i}$.*
- $E' \stackrel{def}{=} (E_\mathfrak{g} - E_\mathfrak{h}) + E_\mathfrak{i}$.
- $R' \stackrel{def}{=} R_\mathfrak{g}$.
- $L'(\alpha) \stackrel{def}{=} \begin{cases} L_\mathfrak{g}(\alpha) & \textit{if } \alpha \in N_\mathfrak{g} \setminus \widehat{N}_\mathfrak{i} \\ L_\mathfrak{i}(\alpha) & \textit{if } \alpha \in \widehat{N}_\mathfrak{i} \end{cases}$ *for all $\alpha \in N' \setminus R'$.*

Example 7. Let $\mathfrak{i}'$ be the $\mathcal{L}$-graph with root (α, β) defined below. Using the $\mathcal{L}$-graphs $\mathfrak{g}$ and $\mathfrak{h}$ from Examples 2 and 5, we get the following $\mathcal{L}$-graph $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}'\}$ (the edge (α, β) occurs twice because it occurs both in $E_{\mathfrak{i}'}$ and in $E_\mathfrak{g} - E_\mathfrak{h}$):

$\mathfrak{i}' =$


$\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}'\} =$


The notation $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}\}$ is extended to the case where $pr(\mathfrak{i}) = pr(\mathfrak{h})$ as follows: $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}\} \stackrel{def}{=} \mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}'\}$, where $\mathfrak{i}'$ is any $\mathcal{L}$-graph substitutable for $\mathfrak{h}$ in $\mathfrak{g}$ such that $\mathfrak{i} \equiv \mathfrak{i}'$. Thus the replacement operation possibly involves a renaming step, to avoid conflicts on the names of the nodes. The next proposition states a straightforward property of subgraph replacement:

Proposition 8. *Let $\mathfrak{g}, \mathfrak{h}, \mathfrak{i}, \mathfrak{j}$ be $\mathcal{L}$-graphs, where $\mathfrak{i} \leq^g \mathfrak{h} \leq^g \mathfrak{g}$ and $pr(\mathfrak{i}) = pr(\mathfrak{j})$. Then $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{h}\{\mathfrak{i} \leftarrow \mathfrak{j}\}\} \equiv \mathfrak{g}\{\mathfrak{i} \leftarrow \mathfrak{j}\}$.*

Remark 9. Note that Proposition 8 would not hold if edges were defined as sets and not as multisets. For instance, consider $\mathcal{L}$-graphs $\mathfrak{g}, \mathfrak{h}$ with two root nodes ρ_1, ρ_2, where $\mathfrak{g}$ contains an edge (ρ_1, ρ_2) and $\mathfrak{h}$ contains no edges. If edges are taken as sets then we get $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{g}\} = \mathfrak{g}$ and $\mathfrak{g}\{\mathfrak{g} \leftarrow \mathfrak{h}\} = \mathfrak{h}$, whereas $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{h}\} = \mathfrak{g}$. In our previous work [14], this problem was overcome by restricting ourselves to induced subgraphs (which prevents the replacement of $\mathfrak{h}$ by $\mathfrak{g}$ in $\mathfrak{g}$), but this causes a combinatorial explosion in the definition of the calculus: when one "merges" two subgraphs, it is necessary to add every possible combination of edges connecting a root of the first $\mathcal{L}$-graph to a root of the second one, yielding exponentially many solutions w.r.t. the number of roots (see [14, Definition 30]). Such a behavior is avoided in the new framework.

We now introduce a notion of orthogonality between graphs. The intuition is that two $\mathcal{L}$-graphs will be considered orthogonal if they share no edges and no nodes other than roots.

Definition 10 (Orthogonal graphs). *Let $\mathfrak{g}$ be an $\mathcal{L}$-graph. Two subgraphs $\mathfrak{h}$ and $\mathfrak{i}$ of $\mathfrak{g}$ are* orthogonal *in $\mathfrak{g}$, or simply* orthogonal*, if $\widehat{N}_{\mathfrak{h}} \cap \widehat{N}_{\mathfrak{i}} = \emptyset$ and $E_{\mathfrak{h}} + E_{\mathfrak{i}} \sqsubseteq E_{\mathfrak{g}}$.*

Note that $\mathfrak{h}$ and $\mathfrak{i}$ may share root nodes. Proposition 11 states that the result of the replacement of two orthogonal subgraphs does not depend on the order in which the $\mathcal{L}$-graphs are considered.

Proposition 11. *Let $\mathfrak{g}$ be an $\mathcal{L}$-graph, and let $\mathfrak{h}_1$, $\mathfrak{h}_2$ be orthogonal subgraphs of $\mathfrak{g}$. For all $\mathcal{L}$-graphs $\mathfrak{i}_1, \mathfrak{i}_2$ of respective profiles $pr(\mathfrak{h}_1)$ and $pr(\mathfrak{h}_2)$, $\mathfrak{h}_2$ and $\mathfrak{h}_1$ are subgraphs of $\mathfrak{g}\{\mathfrak{h}_1 \leftarrow \mathfrak{i}_1\}$ and $\mathfrak{g}\{\mathfrak{h}_2 \leftarrow \mathfrak{i}_2\}$, respectively, and $\mathfrak{g}\{\mathfrak{h}_1 \leftarrow \mathfrak{i}_1\}\{\mathfrak{h}_2 \leftarrow \mathfrak{i}_2\} \equiv \mathfrak{g}\{\mathfrak{h}_2 \leftarrow \mathfrak{i}_2\}\{\mathfrak{h}_1 \leftarrow \mathfrak{i}_1\}$.*

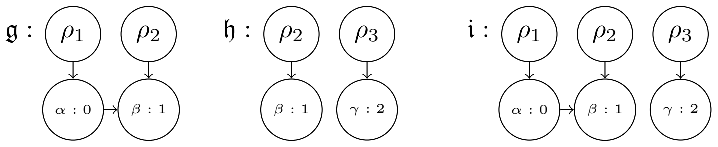
2.2 Graph Merging

Intuitively, a merge of two $\mathcal{L}$-graphs $\mathfrak{g}_1$ and $\mathfrak{g}_2$ denotes any minimal $\mathcal{L}$-graph containing all vertices, labels and edges in $\mathfrak{g}_1$ and $\mathfrak{g}_2$. More formally:

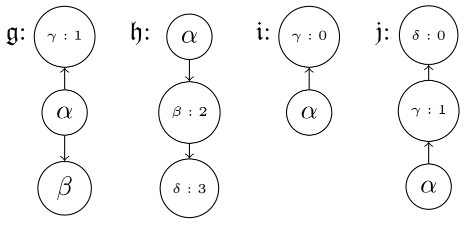
Definition 12. *A* merge *of two $\mathcal{L}$-graphs $\mathfrak{g}_1$ and $\mathfrak{g}_2$ is an $\mathcal{L}$-graph $\mathfrak{h}$ such that: (i) $\mathfrak{g}_i \leq^g \mathfrak{h}$, for all $i = 1, 2$; (ii) $N_{\mathfrak{h}} = N_{\mathfrak{g}_1} \cup N_{\mathfrak{g}_2}$, $E_{\mathfrak{h}} = E_{\mathfrak{g}_1} \sqcup E_{\mathfrak{g}_2}$ and $\widehat{N}_{\mathfrak{h}} = \widehat{N}_{\mathfrak{g}_1} \cup \widehat{N}_{\mathfrak{g}_2}$; (iii) for all $i = 1, 2$ and for all $\alpha \in \widehat{N}_{\mathfrak{g}_i}$, $L_{\mathfrak{h}}(\alpha) = L_{\mathfrak{g}_i}(\alpha)$.*

Note that in contrast to [14, Definition 30], the merge contains no node and edge other than those occurring in $\mathfrak{g}_1$ or $\mathfrak{g}_2$. Moreover, the multiplicity of edges is minimal ($E_{\mathfrak{h}}$ is defined as $E_{\mathfrak{g}_1} \sqcup E_{\mathfrak{g}_2}$ instead of $E_{\mathfrak{g}_1} + E_{\mathfrak{g}_2}$). It is easy to check that a merge of $\mathfrak{g}_1$, $\mathfrak{g}_2$ exists iff $L_{\mathfrak{g}_1}(\alpha) = L_{\mathfrak{g}_2}(\alpha)$ holds for all $\alpha \in \widehat{N}_{\mathfrak{g}_1} \cap \widehat{N}_{\mathfrak{g}_2}$. Moreover, all the merges are equal up to a permutation of their roots.

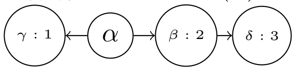
Example 13. Consider the following $\mathcal{L}$-graphs $\mathfrak{g}$ and $\mathfrak{h}$ below of respective roots (ρ_1, ρ_2) and (ρ_2, ρ_3), where the nodes α, β, γ are labeled by 0, 1 and 2, respectively. These $\mathcal{L}$-graphs admit the following merge $\mathfrak{i}$, of root (ρ_1, ρ_2, ρ_3):



Example 14. Let $\mathfrak{g}$, $\mathfrak{h}$, $\mathfrak{i}$ and $\mathfrak{j}$ be the $\mathcal{L}$-graphs, defined as follows:



The $\mathcal{L}$-graph $\mathfrak{g}$ has roots (α, β) and $\mathfrak{h}, \mathfrak{i}, \mathfrak{j}$ have roots (α). Then $\mathfrak{g}$ and $\mathfrak{h}$ admit the following merge, of root (α):



In contrast, $\mathfrak{g}$ and $\mathfrak{i}$ admit no merge (since γ has different labels in the two graphs), and neither do $\mathfrak{g}$ and $\mathfrak{j}$ (due to the edge connecting the non-root node γ to δ, that is outside of $\mathfrak{g}$).

Lemma 15. *Let $\mathfrak{g}$ be an $\mathcal{L}$-graph and let $\mathfrak{h}, \mathfrak{i}$ be subgraphs of $\mathfrak{g}$. Then $\mathfrak{h}$ and $\mathfrak{i}$ admit a merge $\mathfrak{j}$, and for all merges $\mathfrak{j}$ of $\mathfrak{h}$ and $\mathfrak{i}$ we have $\mathfrak{j} \leq^g \mathfrak{g}$.*

3 An Equational Logic on Graphs

We now define equational clauses built on $\mathcal{L}$-graphs and their semantics.

Definition 16. *An* equation *is an unordered pair written $\mathfrak{g} \approx \mathfrak{h}$, where $\mathfrak{g}, \mathfrak{h}$ are $\mathcal{L}$-graphs such that $R_\mathfrak{g} = R_\mathfrak{h}$. A* literal *is either an equation (positive literal) or the negation of an equation, written $\mathfrak{g} \not\approx \mathfrak{h}$ (negative literal). A* clause *is a disjunction of literals. The disjunction may be empty, in which case the clause is written $\Box$. A clause is* Horn *if it contains at most one positive literal. A set of clauses is* Horn *if it contains only Horn clauses.*

Note that we assume for technical convenience that the two members of an equation share the same roots. $\mathcal{N}$-renamings μ are extended to equations, literals and clauses in a straightforward way: $\mu(\mathfrak{g} \approx \mathfrak{h}) \overset{def}{=} \mu(\mathfrak{g}) \approx \mu(\mathfrak{h})$, $\mu(\mathfrak{g} \not\approx \mathfrak{h}) \overset{def}{=} \mu(\mathfrak{g}) \not\approx \mu(\mathfrak{h})$ and $\mu(C \vee D) \overset{def}{=} \mu(C) \vee \mu(D)$. The relation $\equiv$ is extended accordingly.

Sets of clauses built on $\mathcal{L}$-graphs will be interpreted w.r.t. a congruence on $\mathcal{L}$-graphs. Graph congruences are defined in same way as for terms, except that we also assume that they are closed under isomorphism.

Definition 17 (Graph Congruence). *A binary relation $\bowtie$ on $\mathcal{L}$-graphs is* closed under isomorphisms *if $\mathfrak{i} \bowtie \mathfrak{h}$ when $\mathfrak{g} \bowtie \mathfrak{h}$ and $\mathfrak{g} \equiv \mathfrak{i}$. It is* closed under embeddings *if $\mathfrak{h} \bowtie \mathfrak{i}$ entails $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}\} \bowtie \mathfrak{g}$. A* congruence *is an equivalence relation on $\mathcal{L}$-graphs that is closed under isomorphisms and embeddings.*

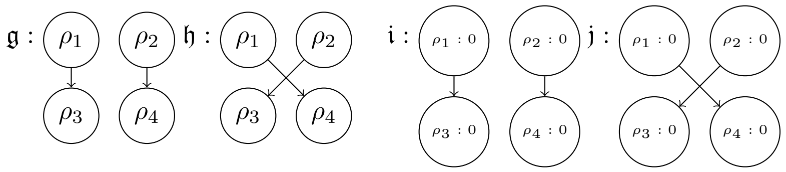
Definition 18. *A congruence $\sim$* validates *an expression E (written $\sim \models E$) iff one of the following conditions holds: (i) E is an equation $\mathfrak{g} \approx \mathfrak{h}$ and $\mathfrak{g} \sim \mathfrak{h}$; (ii) E is a literal $\mathfrak{g} \not\approx \mathfrak{h}$ and $\mathfrak{g} \not\sim \mathfrak{h}$; (iii) E is a clause C and $\sim$ validates at least one literal in C; (iv) E is a set of clauses Γ and $\sim$ validates all the clauses in Γ. A congruence $\sim$ is a* model *of E if $\sim \models E$. An expression is* satisfiable *if it admits a model and* unsatisfiable *otherwise. A* tautology *is a clause that is true in all congruences.*

4 Superposition Calculus with Uninterpreted Labels

We define a superposition calculus for testing the satisfiability of sets of clauses. This calculus is *strict* (see, e.g., [2]) in the sense that it does not use the equational factorization rule (as defined in [1]), but uses instead the standard factorization rule that unifies both members of two equations. This choice is motivated by the fact that, as shown in Example 22, graph superposition is not compatible with tautology deletion (except when the clauses are Horn). Since tautology deletion is disabled for non-Horn clauses, equational factorization is not needed anyway. Selection functions are not considered, since they are not compatible with the redundancy criterion.

The usual superposition calculus [1] is parameterized by a *reduction order*, i.e., an order on terms that is well-founded, total on ground terms, and closed under substitutions and embeddings. In the case of $\mathcal{L}$-graphs, no such order possibly exists, if we also add the natural requirement that the order must be closed under renamings, as evidenced by the following example:

Example 19. Assume that an order $<$ exists, satisfying the following properties: $<$ is well-founded, closed under isomorphisms and embeddings, and total up to isomorphism (i.e., if $\mathfrak{g} \not\equiv \mathfrak{h}$ then either $\mathfrak{g} < \mathfrak{h}$ or $\mathfrak{h} < \mathfrak{g}$). Consider the $\mathcal{L}$-graphs $\mathfrak{g}$ and $\mathfrak{h}$ with roots $(\rho_1, \rho_2, \rho_3, \rho_4)$ and containing no labels, as well as the $\mathcal{L}$-graphs $\mathfrak{i}, \mathfrak{j}$ with an empty sequence of roots, where all nodes are labeled by 0:



It is clear that $\mathfrak{g} \not\equiv \mathfrak{h}$. Indeed, if $\mu(\mathfrak{g}) = \mathfrak{h}$ holds for some $\mathcal{N}$-renaming μ, then $\mu(R_\mathfrak{g}) = R_\mathfrak{h}$, i.e., $\mu((\rho_1, \rho_2, \rho_3, \rho_4)) = (\rho_1, \rho_2, \rho_3, \rho_4)$, which entails that μ is the identity on these nodes. Thus we cannot have $\mu(E_\mathfrak{g}) = E_\mathfrak{h}$, as the first root (ρ_1) is connected to the third root (ρ_3) in $\mathfrak{g}$ and to the fourth one (ρ_4) in $\mathfrak{h}$. Consequently, we have either $\mathfrak{g} < \mathfrak{h}$ or $\mathfrak{h} < \mathfrak{g}$. Now we also have $\mathfrak{g} \leq^g \mathfrak{i}$ and $\mathfrak{h} \leq^g \mathfrak{j}$, and it is easy to check that $\mathfrak{i}\{\mathfrak{g} \leftarrow \mathfrak{h}\} = \mathfrak{j}$ and $\mathfrak{j}\{\mathfrak{h} \leftarrow \mathfrak{g}\} = \mathfrak{i}$. Thus we have either $\mathfrak{i} < \mathfrak{j}$ or $\mathfrak{j} < \mathfrak{i}$. But since $R_\mathfrak{i} = R_\mathfrak{j} = ()$ we have $\mathfrak{i} \equiv \mathfrak{j}$: indeed, if $\mu(\rho_1) = \rho_1$, $\mu(\rho_2) = \rho_2$, $\mu(\rho_3) = \rho_4$ and $\mu(\rho_4) = \rho_3$, then $\mu(\mathfrak{i}) = \mathfrak{j}$.

We thus slightly relax the requirement of having a reduction order, and consider instead a pre-order $<$ on $\mathcal{L}$-graphs, that is well-founded, closed under isomorphisms and embeddings, and contains $\leq^g$. We write $\mathfrak{g} < \mathfrak{h}$ if $\mathfrak{g} \leq \mathfrak{h}$ and $\mathfrak{h} \not\leq \mathfrak{g}$, and we write $\mathfrak{g} \simeq \mathfrak{h}$ if $\mathfrak{g} \leq \mathfrak{h}$ and $\mathfrak{h} \leq \mathfrak{g}$. We also assume that the equivalence classes of $\simeq$ are finite, up to isomorphism. It is clear that such pre-orders exist, for instance, the pre-order: $\mathfrak{g} \leq \mathfrak{h} \iff \mathit{card}(N_\mathfrak{g}) \leq \mathit{card}(N_\mathfrak{h})$ fulfills the above properties.

Similarly to the usual superposition calculus, we associate every literal L with a multiset defined as follows: $\mathit{mset}(\mathfrak{g} \not\approx \mathfrak{h}) \overset{\mathit{def}}{=} \{\{\mathfrak{g}, \mathfrak{h}\}\}$ and $\mathit{mset}(\mathfrak{g} \approx \mathfrak{h}) \overset{\mathit{def}}{=} \{\{\mathfrak{g}\}, \{\mathfrak{h}\}\}$. For every clause $C = L_1 \vee \cdots \vee L_n$, we define: $\mathit{mset}(C) \overset{\mathit{def}}{=} \{\mathit{mset}(L_i) \mid i = 1, \ldots, n\}$. Any order or preorder $\rhd$ on $\mathcal{L}$-graphs may then be extended into an order on clauses as follows: $C \rhd D \iff \mathit{mset}(C) \rhd_m \mathit{mset}(D)$, where $\rhd_m$ denotes the multiset extension of $\rhd$ (note that $\rhd_m$ is also a (pre)order). A literal L is $<$*-maximal* in a clause C if there is no literal $L' \in C$ such that $L' > L$. An $\mathcal{L}$-graph $\mathfrak{g}$ is $<$*-maximal* in a literal L if L contains no $\mathcal{L}$-graph $\mathfrak{g}'$ such that $\mathfrak{g}' > \mathfrak{g}$.

A literal L is *eligible* in a clause C if L is a $<$-maximal literal in C. Intuitively, eligible literals are those that may be considered for performing inferences. For instance, given a clause $(\mathfrak{g} \approx \mathfrak{h}) \vee (\mathfrak{i} \approx \mathfrak{j})$, if $(\mathfrak{g} \approx \mathfrak{h}) > (\mathfrak{i} \approx \mathfrak{j})$, then $\mathfrak{g} \approx \mathfrak{h}$ is eligible but not $\mathfrak{i} \approx \mathfrak{j}$. Consequently the inference rules (as defined in Section 4.1) will be allowed to replace $\mathfrak{g}$ by $\mathfrak{h}$ using the equation $\mathfrak{g} \approx \mathfrak{h}$ (provided $\mathfrak{g} \not< \mathfrak{h}$) but not, e.g., $\mathfrak{i}$ by $\mathfrak{j}$ (this restricts the number of inferences and prune the search space). Non eligible literals are simply attached to the conclusion of the inference but they play no active role until they (eventually) become eligible.

4.1 Inference Rules

The Superposition calculus SC is defined by the following rules: $\texttt{Sp}^+$ (positive superposition), $\texttt{Sp}^-$ (negative superposition), R (Reflection) and F (Factoring). The rules and their side conditions are very similar to those of the usual (ground) superposition calculus, except for the use of the merging operation for positive superposition. To simplify notations, the rules are defined modulo isomorphims, which means that one has to find a renaming of the premises such that the considered rule applies (this can be done using standard algorithms for finding graph homomorphisms). For instance, with this convention, the Reflection rule R actually removes all equations of the form $\mathfrak{g} \not\approx \mathfrak{h}$, with $\mathfrak{g} \equiv \mathfrak{h}$.

$$\texttt{Sp}^+ : \frac{\mathfrak{g}_1 \approx \mathfrak{h}_1 \vee C_1 \quad \mathfrak{g}_2 \approx \mathfrak{h}_2 \vee C_2}{\mathfrak{i}\{\mathfrak{g}_1 \leftarrow \mathfrak{h}_1\} \approx \mathfrak{i}\{\mathfrak{g}_2 \leftarrow \mathfrak{h}_2\} \vee C_1 \vee C_2}$$

where:

1. $\mathfrak{i}$ is a merge of $\mathfrak{g}_1$ and $\mathfrak{g}_2$, and $\mathfrak{g}_1, \mathfrak{g}_2$ are not orthogonal;
2. $\mathfrak{g}_i \approx \mathfrak{h}_i$ is eligible in $\mathfrak{g}_i \approx \mathfrak{h}_i \vee C_i$ for $i = 1, 2$.
3. $\mathfrak{g}_i \not< \mathfrak{h}_i$ for $i = 1, 2$.

The non-orthogonality condition is the analogous of the non-variable condition of the usual calculus, it dismisses trivial replacements.

$$\texttt{Sp}^- : \frac{\mathfrak{g} \approx \mathfrak{h} \vee C \quad \mathfrak{i} \not\approx \mathfrak{j} \vee D}{\mathfrak{i}\{\mathfrak{g} \leftarrow \mathfrak{h}\} \not\approx \mathfrak{j} \vee C \vee D}$$

where:

1. $\mathfrak{g} \leq^g \mathfrak{i}$;
2. $\mathfrak{g} \approx \mathfrak{h}$ and $\mathfrak{i} \not\approx \mathfrak{j}$ are eligible in $\mathfrak{g} \approx \mathfrak{h} \vee C$ and $\mathfrak{i} \not\approx \mathfrak{j} \vee D$, respectively.
3. $\mathfrak{g} \not< \mathfrak{h}$ and $\mathfrak{i} \not< \mathfrak{j}$.

$$\texttt{F} : \frac{\mathfrak{g} \approx \mathfrak{h} \vee \mathfrak{g} \approx \mathfrak{h} \vee C}{\mathfrak{g} \approx \mathfrak{h} \vee C} \qquad \text{if } \mathfrak{g} \approx \mathfrak{h} \text{ is eligible in } \mathfrak{g} \approx \mathfrak{h} \vee \mathfrak{g} \approx \mathfrak{h} \vee C.$$

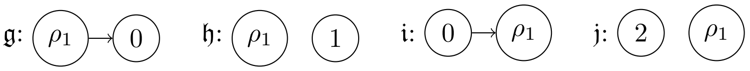
$$\texttt{R} : \frac{\mathfrak{g} \not\approx \mathfrak{g} \vee C}{C} \qquad \text{if } \mathfrak{g} \not\approx \mathfrak{g} \text{ is eligible in } \mathfrak{g} \not\approx \mathfrak{g} \vee C.$$

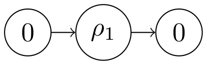
Lemma 20. *The rules* $\texttt{Sp}^+$, $\texttt{Sp}^-$, F *and* R *are sound, i.e., for all congruences* $\sim$ *and for all clauses* C *deducible from a set of premises* Γ, *we have* $\sim\models \Gamma \implies \sim\models C$.

4.2 Redundancy

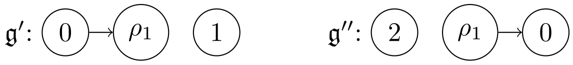
In the usual superposition calculus [1], a clause is redundant if all its ground instances are entailed by smaller clauses (w.r.t. the considered order). Such clauses can be deleted without threatening refutational completeness, which reduces the search space. In our context, such a definition cannot be used, because one of the inference rules –namely $\mathtt{Sp}^+$– may generate clauses that are strictly larger than the premises (hence such clauses would be considered as redundant if the usual criterion were to be used).

Example 21. Consider the clauses: $\mathfrak{g} \approx \mathfrak{h}$ and $\mathfrak{i} \approx \mathfrak{j}$, where $\mathfrak{g}, \mathfrak{h}, \mathfrak{i}, \mathfrak{j}$ are $\mathcal{L}$-graphs with root (ρ_1) that are defined as follows:



The $\mathcal{L}$-graphs $\mathfrak{g}$ and $\mathfrak{i}$ admit the following merge (of root (ρ_1)): 

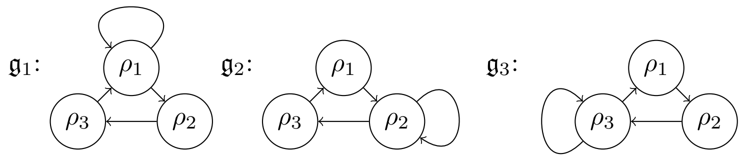
Therefore, rule $\mathtt{Sp}^+$ applies, yielding $\mathfrak{g}' \approx \mathfrak{g}''$, where:



If $\mathcal{L}$-graphs are ordered according to their number of nodes, then we have $(\mathfrak{g}' \approx \mathfrak{g}'') > (\mathfrak{g} \approx \mathfrak{h})$ and $(\mathfrak{g}' \approx \mathfrak{g}'') > (\mathfrak{i} \approx \mathfrak{j})$.

Worse, the calculus is actually incomplete if tautologies are deleted, as shown in the following example.

Example 22. Consider the $\mathcal{L}$-graphs $\mathfrak{g}_1, \mathfrak{g}_2$ and $\mathfrak{g}_3$ with roots (ρ_1, ρ_2, ρ_3):



Let $\dot{\mathfrak{g}}_i$ denote the graph obtained from $\mathfrak{g}_i$ by adding one additional non root node α distinct from ρ_1, ρ_2, ρ_3, with some arbitrary (but fixed) label, e.g., 0. Assume that the graphs are ordered by the number of nodes, so that $\dot{\mathfrak{g}}_i > \mathfrak{g}_j$, $\dot{\mathfrak{g}}_i \simeq \dot{\mathfrak{g}}_j$ and $\mathfrak{g}_i \simeq \mathfrak{g}_j$ (for all $i, j \in \{1, 2, 3\}$). Let $\Gamma = \{\dot{\mathfrak{g}}_1 \approx \mathfrak{g}_2 \vee \dot{\mathfrak{g}}_2 \approx \mathfrak{g}_3 \vee \dot{\mathfrak{g}}_3 \approx \mathfrak{g}_1, \dot{\mathfrak{g}}_1 \not\approx \mathfrak{g}_2 \vee \dot{\mathfrak{g}}_2 \not\approx \mathfrak{g}_3 \vee \dot{\mathfrak{g}}_3 \not\approx \mathfrak{g}_1\}$. Intuitively, every equation $\dot{\mathfrak{g}}_i \approx \mathfrak{g}_j$ where $(i, j) \in \{(1, 2), (2, 3), (3, 1)\}$ states that the semantics of the graph is preserved when the isolated node is deleted and the graph is rotated by 90 degrees clockwise, for each possible position of the loop. Since the graphs are invariant by rotation, all these transformations are actually equivalent. It is easy to check that every clause that can be generated from Γ by applying the negative superposition rule from the first clause into the second clause contains two complementary literals (i.e. two literals of the form $\dot{\mathfrak{g}}_i \approx \mathfrak{g}_j$ and $\dot{\mathfrak{g}}_i \not\approx \mathfrak{g}_j$) hence is a tautology. Moreover, the clauses obtained by superposition using the first clause only either are subsumed by the first clause (if the superposition rule is applied on two different literals) or contains a literal $\mathfrak{g}_i \approx \mathfrak{g}_i$ (hence is a tautology). The equational factorization rule (as defined in [1]) does not apply since $\dot{\mathfrak{g}}_i$ and $\dot{\mathfrak{g}}_j$ are not isomorphic if $i \neq j$. However, consider the $\mathcal{L}$-graphs $\mathfrak{g}'_i, \dot{\mathfrak{g}}'_i$ which contain the same nodes and edges as $\mathfrak{g}_i$ and $\dot{\mathfrak{g}}_i$ respectively, but with roots (ρ_2, ρ_3, ρ_1). It is clear that $\mathfrak{g}'_2 \equiv \mathfrak{g}_1$ and $\mathfrak{g}'_3 \equiv \mathfrak{g}_2$, so that $\dot{\mathfrak{g}}_1 \approx \mathfrak{g}_2 \models \dot{\mathfrak{g}}'_2 \approx \mathfrak{g}'_3$. However, $\dot{\mathfrak{g}}'_2 \leq^g \dot{\mathfrak{g}}_2$ and $\dot{\mathfrak{g}}_2\{\dot{\mathfrak{g}}'_2 \leftarrow \mathfrak{g}'_3\} = \mathfrak{g}_3$, thus $\dot{\mathfrak{g}}_1 \approx \mathfrak{g}_2 \models \dot{\mathfrak{g}}_2 \approx \mathfrak{g}_3$. By a similar reasoning, we may show that $\dot{\mathfrak{g}}_2 \approx \mathfrak{g}_3 \models \dot{\mathfrak{g}}_3 \approx \mathfrak{g}_1$ and $\dot{\mathfrak{g}}_3 \approx \mathfrak{g}_1 \models \dot{\mathfrak{g}}_1 \approx \mathfrak{g}_2$, so that the equations $\dot{\mathfrak{g}}_1 \approx \mathfrak{g}_2$, $\dot{\mathfrak{g}}_2 \approx \mathfrak{g}_3$, and $\dot{\mathfrak{g}}_3 \approx \mathfrak{g}_1$, are actually pairwise equivalent, which entails that Γ is unsatisfiable. However, $\Box$ cannot be derived from Γ if the clauses containing complementary literals are discarded.

Thus, the conditions that ensure that a clause is redundant must be stronger than those of the usual superposition calculus. The definition proposed below covers usual deletion rules such as subsumption. Actually, two different criteria will be used, namely non-strict and strict redundancy, depending on whether the considered clauses are Horn or not. Indeed, in the former case a slightly less restrictive definition can be used, which permits the deletion of (some) tautological clauses.

Definition 23. *Let C, D be two clauses and let Γ be a set of clauses. We say that C is* subsumed *by D and write $C \geq^{sub} D$ if $C = D \vee C'$, up to associativity and commutativity of $\vee$ and isomorphism. We write $C \to_\Gamma D$ (C* demodulates *to D w.r.t. Γ) if C is of the form $\mathfrak{g} \bowtie \mathfrak{h} \vee E$ (with $\bowtie \in \{\approx, \not\approx\}$), $D = \mathfrak{g}\{\mathfrak{i} \leftarrow \mathfrak{j}\} \bowtie \mathfrak{h} \vee E$, and there exists a clause $F \in \Gamma$ such that $F = (\mathfrak{i} \approx \mathfrak{j}) \vee F'$, with $F' \leq^{sub} E$, $\mathfrak{i} > \mathfrak{j}$, $F' < (\mathfrak{i} \approx \mathfrak{j})$ and $(\mathfrak{i} \approx \mathfrak{j}) < (\mathfrak{g} \bowtie \mathfrak{h})$.*

The set of clauses that are redundant w.r.t. a set of clauses Γ is defined inductively as follows. A clause C is redundant *w.r.t. Γ iff one of the following conditions holds: (1) C contains two literals $\mathfrak{g}_1 \approx \mathfrak{g}_2$ and $\mathfrak{g}'_1 \not\approx \mathfrak{g}'_2$, with $\mathfrak{g}_i \equiv \mathfrak{g}'_i$ for $i = 1, 2$; (2) C contains a literal of the form $\mathfrak{g} \approx \mathfrak{h}$ with $\mathfrak{g} \equiv \mathfrak{h}$; (3) $C \geq^{sub} D$, for some $D \in \Gamma$; (4) $C \to_\Gamma D$ and D is redundant. The set of* strictly redundant *ground clauses is defined in a similar way, except that Item 1 is removed.*

Intuitively, the conditions ensuring that C demodulates to D in Definition 23 are meant to ensure that D may be deduced from C by applying the rule $\mathtt{Sp}^+$ or $\mathtt{Sp}^-$ using the clause F (with $D < C$ and $F < C$) and that $\{D\} \cup \Gamma$ is equivalent to $\{C\} \cup \Gamma$. In particular, the condition $F' \leq^{sub} E$ ensures that all the literals added by the inference already occur in C.

Definition 24. *A set of clauses Γ is* saturated *(resp.* strictly saturated*) if every clause that can be deduced from premises in Γ using one of the rules of* SC *(in one step) is redundant (resp. strictly redundant) w.r.t. Γ.*

We prove that SC is refutationally complete. We actually establish two completeness results, the first one for general clauses and the second one for Horn clauses. The latter is stronger since it uses the weaker non-strict saturatedness criterion instead of strict saturatedness.

Theorem 25. *Let Γ be a set of clauses. If $\Box \notin \Gamma$ and Γ is strictly saturated or both Horn and saturated then Γ is satisfiable.*

5 A Constrained Graph Superposition Calculus

We now lift the calculus SC defined in Section 4 into a constrained calculus. The goal is to handle graphs labeled by terms interpreted in some arbitrary theory, and possibly containing variables. To this aim, we attach constraints to the clauses, which are formulas interpreted in the considered theory, asserting conditions on the labels. Such constraints will be updated when inference rules will be applied, by asserting the conditions that are required by the rule applications.

5.1 Constrained Clauses

Let $\mathcal{V}$ be a countably infinite set of *variables* and let Σ be a set of *function symbols*[1]. Each symbol f in Σ is associated with a unique *arity* $\#(f)$. We denote by $\mathcal{T}$ the set of *terms* built inductively as usual on $\mathcal{V}$ and Σ, and by $\mathcal{C}$ the set of first-order formulas, called *constraints*, built inductively as usual on atoms of the form $t \doteq s$, where $t, s \in \mathcal{T}$ using the logical connectives $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$, the quantifiers $\exists, \forall$ and two logical constants $\bot$ and $\top$.

A *substitution* σ is a function mapping all variables x to a term $x\sigma$. The *domain* $dom(\sigma)$ of σ is the set of variables x such that $x\sigma \neq x$. For every term or formula e, we denote by $e\sigma$ the term or formula obtained from e by replacing every (free) variable x by $x\sigma$. A term is *ground* if it contains no variables, and a substitution σ is *ground* if $x\sigma$ is ground for all $x \in dom(\sigma)$.

$\mathcal{T}$*-graph*s are $\mathcal{L}$-graphs with labels in $\mathcal{T}$. A $\mathcal{T}$*-clause* is a clause defined on $\mathcal{T}$-graphs. Substitutions are extended to $\mathcal{T}$-graphs and $\mathcal{T}$-clauses as follows. For every $\mathcal{T}$-graph $\mathfrak{g}$, we denote by $\mathfrak{g}\sigma$ the $\mathcal{T}$-graph such that: $F_{\mathfrak{g}\sigma} = F_{\mathfrak{g}}$ for all $F \in \{N, E, R\}$ and $L_{\mathfrak{g}\sigma}(\alpha) = L_{\mathfrak{g}}(\alpha)\sigma$, for all $\alpha \in \widehat{N}_{\mathfrak{g}}$. Then: $(\mathfrak{g} \approx \mathfrak{h})\sigma \overset{def}{=} \mathfrak{g}\sigma \approx \mathfrak{h}\sigma$, $(\mathfrak{g} \not\approx \mathfrak{h})\sigma \overset{def}{=} \mathfrak{g}\sigma \not\approx \mathfrak{h}\sigma$ and $(C \vee D)\sigma \overset{def}{=} C\sigma \vee D\sigma$. A $\mathcal{T}$-graph $\mathfrak{g}$ is *ground* if for all $\alpha \in \widehat{N}_{\mathfrak{g}}$, $L_{\mathfrak{g}}(\alpha)$ is ground. A $\mathcal{T}$-clause is *ground* if all the $\mathcal{T}$-graphs occurring in it are ground. For every expression (term, $\mathcal{T}$-graph, constraint or $\mathcal{T}$-clause) E, we denote by $\mathcal{V}(E)$ the set of variables (freely) occurring in E.

Definition 26. *A* constrained clause *(or c-clause) is a pair $[C \mid \phi]$, where C is a $\mathcal{T}$-clause and $\phi \in \mathcal{C}$.*

Let $\mathcal{I}$ be some fixed set of first-order interpretations on the signature Σ. For all $I \in \mathcal{I}$, we denote by $dom(I)$ the domain of I and by f^I the interpretation of the function f (with $f \in \Sigma$). For every ground term t and for all $I \in \mathcal{I}$, we denote by $[t]^I$ the value of t in I, inductively defined as usual. To simplify

[1] As usual, predicates may be encoded as functions.

notations, we assume that for every $I \in \mathcal{I}$ and for every $e \in dom(I)$, there exists a ground term t such that $[t]^I = e$.

The satisfiability relation $\models$ relating interpretations in $\mathcal{I}$ and constraints in $\mathcal{C}$ is defined as usual, where $\doteq$ is interpreted as the identity, and $\bot$ and $\top$ are interpreted as false and true, respectively. We write $\phi \models_{\mathcal{I}} \psi$ if the implication $I \models \phi\sigma \implies I \models \psi\sigma$ holds for all $I \in \mathcal{I}$ and for all ground substitutions of domain $\mathcal{V}(\phi) \cup \mathcal{V}(\psi)$; and $\phi \equiv_{\mathcal{I}} \psi$ iff $\phi \models_{\mathcal{I}} \psi$ and $\psi \models_{\mathcal{I}} \phi$. For any set of constraints, we write $I \models S$ iff $I \models \phi$ for all $\phi \in S$. For any constraint (or set of constraints) ϕ, if there exists a ground substitution σ with domain $\mathcal{V}(\phi)$ and an interpretation $I \in \mathcal{I}$ such that $I \models \phi\sigma$, then ϕ is *$\mathcal{I}$-satisfiable* (and *$\mathcal{I}$-unsatisfiable* otherwise). For instance, the fixed set of first-order interpretations may be the set $\mathcal{I}_1$ of first-order interpretations on Σ that satisfy the above condition on the domain (this is not restrictive provided there are infinitely many ground terms), in which case $\mathcal{I}$-satisfiability is simply the standard satisfiability in first-order clausal logic, or the set $\mathcal{I}_{\mathbb{N}}$ of interpretations of domain $\mathbb{N}$ interpreting the functions $0, 1, +$ as usual. We say that $\mathcal{I}$ is *compact* if for every $\mathcal{I}$-unsatisfiable set of constraints S there exists a finite set $S' \subseteq S$ such that S' is $\mathcal{I}$-unsatisfiable. It is well-known that $\mathcal{I}_1$ is compact [21] and that $\mathcal{I}_{\mathbb{N}}$ is not compact[2].

Any ground $\mathcal{T}$-graph may be transformed into a $dom(I)$-graph by replacing the labels by their interpretations in I. More formally:

Definition 27. *For all $I \in \mathcal{I}$ and for all ground $\mathcal{T}$-graphs $\mathfrak{g}$ we denote by $[\mathfrak{g}]^I$ the graph such that $F_{[\mathfrak{g}]^I} = F_{\mathfrak{g}}$ for all $F \in \{N, E, R\}$ and $L_{[\mathfrak{g}]^I}(\alpha) = [L_{\mathfrak{g}}(\alpha)]^I$, for all $\alpha \in \widehat{N}_{\mathfrak{g}}$. For every ground $\mathcal{T}$-clause C, we denote by $[C]^I$ the clause obtained from C by replacing every $\mathcal{T}$-graph $\mathfrak{g}$ by $[\mathfrak{g}]^I$. For all sets of c-clauses Γ, we denote by $[\Gamma]^I$ the set of clauses of the form $[C\sigma]^I$, where $C \in \Gamma$ and σ is a substitution mapping every variable in C to a ground term.*

Note that by definition, all the labels of $[\mathfrak{g}]^I$ are elements of the domain of I. Proposition 28 follows immediately from Definition 27.

Proposition 28. *Let $\mathfrak{g}, \mathfrak{h}$ be $\mathcal{T}$-graphs, let $I \in \mathcal{I}$ and let σ be a ground substitution with domain $\mathcal{V}(\mathfrak{g}) \cup \mathcal{V}(\mathfrak{h})$. If $\mathfrak{g} \equiv \mathfrak{h}$ then $[\mathfrak{g}\sigma]^I \equiv [\mathfrak{h}\sigma]^I$.*

Definition 29. *An $\mathcal{I}$-*interpretation* is a pair $(I, \sim)$, where $I \in \mathcal{I}$ and $\sim$ is a congruence on $dom(I)$-graphs. An $\mathcal{I}$-interpretation $(I, \sim)$* validates *a set of c-clauses Γ (written $(I, \sim) \models \Gamma$) if $\sim \models [\Gamma]^I$.*

5.2 Lifting the Calculus

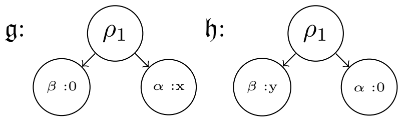
In the constrained calculus, the equality of labels will not be checked when an inference rule is applied. Instead, the corresponding conditions will be extracted from the considered graphs and added to the constraints of the conclusion. We first introduce a relation stating that two $\mathcal{T}$-graphs are identical up to their

[2] For instance, the set $\{n \doteq i \mid i \in \mathbb{N}\}$ is unsatisfiable if n is interpreted as a natural number, but admits no finite unsatisfiable subset.

labels. This relation is parameterized by a constraint that asserts conditions on the labels ensuring that the graphs are identical (modulo $\mathcal{I}$).

Definition 30. *Let $\mathfrak{g}, \mathfrak{h}$ be two $\mathcal{T}$-graphs and let $\phi \in \mathcal{C}$. We write $\mathfrak{g} =_\phi \mathfrak{h}$ if $N_\mathfrak{g} = N_\mathfrak{h}$, $E_\mathfrak{g} = E_\mathfrak{h}$, $R_\mathfrak{g} = R_\mathfrak{h}$, and $\phi = \bigwedge_{\alpha \in \widehat{N}_\mathfrak{g}} (L_\mathfrak{g}(\alpha) \doteq L_\mathfrak{h}(\alpha))$ (up to associativity and commutativity of $\wedge$).*

Example 31. Consider the $\mathcal{T}$-graphs $\mathfrak{g}$ and $\mathfrak{h}$ below, of root (ρ_1). We have $\mathfrak{g} =_\phi \mathfrak{h}$, with $\phi = (x \doteq 0 \wedge 0 \doteq y)$.



Every relation between $\mathcal{T}$-graphs or $\mathcal{T}$-clauses may be adapted in a similar way, keeping the conditions on the nodes, edges and roots, and asserting conditions ensuring that the label of every given node is unique (up to equality modulo $\mathcal{I}$). Definitions 32 and 33 lift the subgraph and subsumption relations, respectively:

Definition 32. *We write $\mathfrak{h} \leq^g_\phi \mathfrak{g}$ if $N_\mathfrak{h} \subseteq N_\mathfrak{g}$; $E_\mathfrak{h} \sqsubseteq E_\mathfrak{g}$; every node $\alpha \in N_\mathfrak{h}$ occurring in $R_\mathfrak{g}$ also occurs in $R_\mathfrak{h}$; if $\alpha \in N_\mathfrak{h}$ occurs in an edge in $E_\mathfrak{g} \setminus E_\mathfrak{h}$ then $\alpha \in R_\mathfrak{h}$, and $\phi = \bigwedge_{\alpha \in \widehat{N}_\mathfrak{h}} L_\mathfrak{h}(\alpha) \doteq L_\mathfrak{g}(\alpha)$. The notation $\mathfrak{g}\{\mathfrak{h} \leftarrow \mathfrak{i}\}$ may be extended to the case where $\mathfrak{h} \leq^g_\phi \mathfrak{g}$ (following Definition 6). Orthogonality is extended accordingly (as it does not depend on labels).*

Definition 33. *We write $C \leq^{sub}_\phi D$ if C and D are respectively of the form (up to associativity and commutativity of $\vee$ and isomorphism): $\bigvee_{i=1}^n \mathfrak{g}_i \bowtie_i \mathfrak{h}_i$, and $\bigvee_{i=1}^n \mathfrak{g}'_i \bowtie_i \mathfrak{h}'_i \vee D'$, with $\mathfrak{g}_i =_{\phi_i} \mathfrak{g}'_i$, $\mathfrak{h}_i =_{\psi_i} \mathfrak{h}'_i$ (for all $i = 1, \dots, n$) and $\phi = \bigwedge_{i=1}^n (\phi_i \wedge \psi_i)$.*

The notion of a merge is extended analogously:

Definition 34. *A* ϕ-merge *of two $\mathcal{T}$-graphs $\mathfrak{g}_1$ and $\mathfrak{g}_2$ is a $\mathcal{T}$-graph $\mathfrak{h}$ such that:*
- *$N_\mathfrak{h} = N_{\mathfrak{g}_1} \cup N_{\mathfrak{g}_2}$, $E_\mathfrak{h} = E_{\mathfrak{g}_1} \sqcup E_{\mathfrak{g}_2}$, and $\widehat{N}_\mathfrak{h} = \widehat{N}_{\mathfrak{g}_1} \cup \widehat{N}_{\mathfrak{g}_2}$.*
- *For every node $\alpha \in \widehat{N}_\mathfrak{h}$, we have $L_\mathfrak{h}(\alpha) = L_{\mathfrak{g}_i}(\alpha)$, for some (arbitrarily chosen) $i = 1, 2$ such that $L_{\mathfrak{g}_i}(\alpha)$ is defined.*
- *$\phi = \bigwedge_{\alpha \in \widehat{N}_{\mathfrak{g}_1} \cap \widehat{N}_{\mathfrak{g}_2}} L_{\mathfrak{g}_1}(\alpha) \doteq L_{\mathfrak{g}_2}(\alpha)$.*

We now lift the order relation. Let $\leq_I$ (for $I \in \mathcal{I}$) be a family of well-founded preorders on $dom(I)$-$\mathcal{T}$-graphs that are closed under isomorphisms and embeddings and contain $\leq^g$. Let $\leq_\phi$ (for $\phi \in \mathcal{C}$) be a family of pre-orders on $\mathcal{T}$-graphs satisfying the following conditions: $\mathfrak{g} >_\phi \mathfrak{h} \implies \mathfrak{g} >_\psi \mathfrak{h}$, for all constraints ϕ, ψ such that $\psi \models_\mathcal{I} \phi$, and $(I \models \phi \wedge \mathfrak{g} >_\phi \mathfrak{h}) \implies [\mathfrak{g}]^I >_I [\mathfrak{h}]^I$. The simplest solution in practice is to order $\mathcal{T}$-graphs according to their number of nodes, in which case the order does not depend on I or ϕ: $\mathfrak{g} \leq_I \mathfrak{h} \iff \mathfrak{g} \leq_\phi$

$\mathfrak{h} \iff \mathit{card}(N_{\mathfrak{g}}) \leq \mathit{card}(N_{\mathfrak{h}})$. However, our framework is meant to be general enough to cope with orders that take labels into account.

A literal L is *maximal* in a c-clause $[C \mid \phi]$ if there is no literal $L' \in C$ such that $L' >_{\phi} L$. It is *eligible* in a c-clause $[C \mid \phi]$ if L is a $>_{\phi}$-maximal literal in C.

We are now in the position to define the constrained inference rules. As for the rules in Section 4.1, they apply modulo isomorphism. We assume as for the standard resolution or superposition calculus that the premises share no variables. In every rule, the conclusion inherits the constraints of the premises together with additional conditions on the labels which makes the inference valid. In all rules, the eligibility condition is tested after adding all the constraints enabling the inference, as this yields the most restrictive condition, thus reducing the branching factor.

$$\mathtt{Sp}^{+} : \frac{[\mathfrak{g}_1 \approx \mathfrak{h}_1 \vee C_1 \mid \phi_1] \quad [\mathfrak{g}_2 \approx \mathfrak{h}_2 \vee C_2 \mid \phi_2]}{[\mathfrak{i}\{\mathfrak{g}_1 \leftarrow \mathfrak{h}_1\} \approx \mathfrak{i}\{\mathfrak{g}_2 \leftarrow \mathfrak{h}_2\} \vee C_1 \vee C_2 \mid \phi_1 \wedge \phi_2 \wedge \psi]}$$

where:

1. $\mathfrak{i}$ is a ψ-merge of $\mathfrak{g}_1$ and $\mathfrak{g}_2$ and $\mathfrak{g}_1$ and $\mathfrak{g}_2$ are not orthogonal;
2. $\mathfrak{g}_i \approx \mathfrak{h}_i$ is eligible in $[\mathfrak{g}_i \approx \mathfrak{h}_i \vee C_i \mid \phi_1 \wedge \phi_2 \wedge \psi]$ (for all $i = 1, 2$);
3. $\mathfrak{g}_i \not<_{\phi_1 \wedge \phi_2 \wedge \psi} \mathfrak{h}_i$ (for all $i = 1, 2$).

$$\mathtt{Sp}^{-} : \frac{[\mathfrak{g} \approx \mathfrak{h} \vee C \mid \phi] \quad [\mathfrak{i} \not\approx \mathfrak{j} \vee D \mid \psi]}{[\mathfrak{i}\{\mathfrak{g} \leftarrow \mathfrak{h}\} \not\approx \mathfrak{j} \vee C \vee D \mid \phi \wedge \psi \wedge \xi]}$$

where:

1. $\mathfrak{g} \leq_{\xi}^{g} \mathfrak{i}$ (note that ξ is uniquely defined by Definition 32);
2. $\mathfrak{g} \approx \mathfrak{h}$ and $\mathfrak{i} \not\approx \mathfrak{j}$ are eligible in $[\mathfrak{g} \approx \mathfrak{h} \vee C \mid \phi \wedge \psi \wedge \xi]$ and $[\mathfrak{i} \not\approx \mathfrak{j} \vee D \mid \phi \wedge \psi \wedge \xi]$, respectively;
3. $\mathfrak{g} \not<_{\phi \wedge \psi \wedge \xi} \mathfrak{h}$ and $\mathfrak{i} \not<_{\phi \wedge \psi \wedge \xi} \mathfrak{j}$.

$$\mathtt{F} : \frac{[\mathfrak{g} \approx \mathfrak{h} \vee \mathfrak{g}' \approx \mathfrak{h}' \vee C \mid \phi]}{[\mathfrak{g} \approx \mathfrak{h} \vee C \mid \phi \wedge \psi \wedge \psi']}$$

where $\mathfrak{g} \approx \mathfrak{h}$ is eligible in $[\mathfrak{g} \approx \mathfrak{h} \vee \mathfrak{g}' \approx \mathfrak{h}' \vee C \mid \phi \wedge \psi \wedge \psi']$, $\mathfrak{g} =_{\psi} \mathfrak{g}'$, and $\mathfrak{h} =_{\psi'} \mathfrak{h}'$.

$$\mathtt{R} : \frac{[\mathfrak{g} \not\approx \mathfrak{h} \vee C \mid \phi]}{[C \mid \phi \wedge \psi]}$$

where $\mathfrak{g} \not\approx \mathfrak{h}$ is eligible in $[\mathfrak{g} \not\approx \mathfrak{h} \vee C \mid \phi \wedge \psi]$ and $\mathfrak{g} =_{\psi} \mathfrak{h}$.

5.3 Soundness and Refutational Completeness

We establish the soundness and completeness of the constrained calculus, by lifting the corresponding properties for the base calculus. Note that semi decidability holds only if the base theory is semi-decidable[3] and compact (otherwise it is easy to see that unsatisfiability is not semi-decidable in general).

[3] in the sense that there exists a semi-decision procedure to check whether a formula in $\mathcal{C}$ is unsatisfiable.

Lemma 35. *The rules* $\mathtt{Sp}^+$, $\mathtt{Sp}^-$, $\mathtt{F}$ *and* $\mathtt{R}$ *(applied on c-clauses) are sound, i.e., for all* $\mathcal{I}$*-interpretations* $(I, \sim)$ *and for all c-clauses* $[C \mid \phi]$ *deducible for a set of premises* Γ*, we have* $(I, \sim) \models \Gamma \implies (I, \sim) \models [C \mid \phi]$.

The redundancy criterion may be lifted as follows:

Definition 36. *A c-clause* $[C \mid \phi]$ *is (strictly)* $\mathcal{I}$-redundant *in a set of c-clauses* Γ *if for all ground substitutions* σ *of domain* $\mathcal{V}(C) \cup \mathcal{V}(\phi)$ *and for all* $I \in \mathcal{I}$ *such that* $I \models \phi\sigma$*, the clause* $[C\sigma]^I$ *is (strictly) redundant in* $[\Gamma]^I$.

A set of c-clauses Γ *is (strictly)* saturated *if every c-clause that is deducible from* Γ *by the rules above is (strictly)* $\mathcal{I}$*-redundant in* Γ.

Theorem 37. *Let* Γ *be a set of c-clauses. If* Γ *is unsatisfiable and strictly saturated or Horn and saturated, then* Γ *contains a set of c-clauses* $\{[\Box \mid \phi_I] \mid I \in \mathcal{I}\}$ *such that for every* $I \in \mathcal{I}$, $I \models \exists \boldsymbol{x}_I.\phi_i$*, with* $\boldsymbol{x}_I = \mathcal{V}(\phi_I)$*. If, moreover,* $\mathcal{I}$ *is compact, then* Γ *contains a finite set of c-clauses* $\{[\Box \mid \phi_i] \mid i = 1, \dots, n\}$ *such that* $\bigwedge_{i=1}^{n} \neg(\exists \boldsymbol{x}.\phi_i)$ *is* $\mathcal{I}$*-unsatisfiable, with* $\boldsymbol{x}_i = \mathcal{V}(\phi_i)$.

5.4 Redundancy Testing

The redundancy criterion in Definition 36 is very general, but it may be difficult to test in practice. We thus introduce a second notion of redundancy, defined directly on constrained clauses, that is stronger and easier to decide.

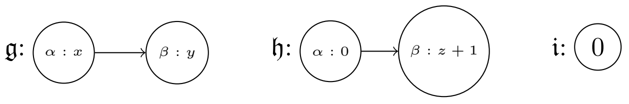
Definition 38. *Let* $[C \mid \phi], [D \mid \psi]$ *be two clauses and let* Γ *be a set of clauses. Let* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *be the vectors of variables occurring in* $[C \mid \phi]$ *and* $[D \mid \psi]$*, respectively (we assume by renaming that* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *share no variable).*

We say that $[C \mid \phi]$ *is* subsumed *by* $[D \mid \psi]$ *and we write* $[C \mid \phi] \geq^{sub} [D \mid \psi]$ *if there exists* $\xi \in \mathcal{C}$ *such that* $D \leq_{\xi}^{sub} C$ *and* $\phi \models_{\mathcal{I}} \exists \boldsymbol{y}.(\psi \wedge \xi)$.

We write $[C \mid \phi] \to_{\Gamma} [D \mid \psi]$ *(*$[C \mid \phi]$ demodulates *to* $[D \mid \psi]$ *w.r.t.* Γ*) if* C *is of the form* $\mathfrak{g} \bowtie \mathfrak{h} \vee E$, $D = \mathfrak{g}\{\mathfrak{i} \leftarrow \mathfrak{j}\} \bowtie \mathfrak{h} \vee E$*, and there exists a c-clause* $[F \mid \xi] \in \Gamma$ *(with free variables* $\boldsymbol{z}$*) such that* $F = (\mathfrak{i} \approx \mathfrak{j}) \vee F'$, $\mathfrak{i} \leq_{\xi'}^{g} \mathfrak{g}$, $F' \leq_{\xi''}^{sub} E$, $\phi \models_{\mathcal{I}} \exists \boldsymbol{y}.\exists \boldsymbol{z}.(\psi \wedge \xi \wedge \xi' \wedge \xi'')$, $\mathfrak{i} >_{\xi} \mathfrak{j}$, $F' <_{\xi} (\mathfrak{i} \approx \mathfrak{j})$ *and* $(\mathfrak{i} \approx \mathfrak{j}) <_{\xi} (\mathfrak{g} \bowtie \mathfrak{h})$.

A c-clause $[C \mid \phi]$ *is* redundant *w.r.t.* Γ *iff one of the following conditions holds: (1)* $\exists \boldsymbol{x}.\phi$ *is* $\mathcal{I}$*-unsatisfiable, with* $\boldsymbol{x} = \mathcal{V}(\phi)$*. (2)* C *contains two literals* $\mathfrak{g}_1 \approx \mathfrak{g}_2$ *and* $\mathfrak{g}'_1 \not\approx \mathfrak{g}'_2$*, with* $\mathfrak{g}_i =_{\phi_i} \mathfrak{g}'_i$*, and* $\phi \models_{\mathcal{I}} \phi_i$ *(for all* $i = 1, 2$*); (3)* C *contains a literal of the form* $\mathfrak{g} \approx \mathfrak{h}$ *with* $\mathfrak{g} =_{\psi} \mathfrak{h}$ *and* $\phi \models_{\mathcal{I}} \psi$*; (4)* $[C \mid \phi] \geq^{sub} [D \mid \psi]$*, for some* $[D \mid \psi] \in \Gamma$*; (5)* $[C \mid \phi] \to_{\Gamma} [D \mid \psi]$ *and* $[D \mid \psi]$ *is redundant. The notion of strictly redundant c-clause is defined in a similar way, removing Item 2.*

Example 39. Consider the following $\mathcal{T}$-graphs, of root ():



We have $\mathfrak{g} \approx \mathfrak{i} \leq_{\phi}^{sub} \mathfrak{h} \approx \mathfrak{i}$, with $\phi = (x \doteq 0 \wedge y \doteq z + 1 \wedge 0 \doteq 0)$. Thus, if $\mathcal{I}$ only contains the standard model of Presburger arithmetic, then $[\mathfrak{g} \approx \mathfrak{i} \mid y \not\approx 0]$ subsumes $[\mathfrak{h} \approx \mathfrak{i} \mid \top]$.

The following lemma states the relation between the new notion of redundancy and $\mathcal{I}$-redundancy (as defined in Definition 36).

Lemma 40. *Let Γ be a set of c-clauses. If $[C \mid \phi]$ is (strictly) redundant w.r.t. Γ then it is (strictly) $\mathcal{I}$-redundant w.r.t. Γ.*

Remark 41. By the previous definitions, checking whether a given c-clause is (strictly) redundant involves testing the validity of entailments of the form $\phi \models_{\mathcal{I}} \exists \boldsymbol{y}.\psi$, which may be infeasible in practice (for instance the problem is undecidable if $\mathcal{I}$ contains all interpretations). Stronger conditions may be used instead, e.g., one may check whether there exists a substitution σ such that ϕ is of the form $\psi\sigma \wedge \psi$, which is decidable.

6 Conclusion

We devised a constrained superposition calculus to test the satisfiability of sets of clauses defined over graphs. Its soundness and refutational completeness was established, modulo a redundancy criterion that captures the usual deletion and simplification rules: subsumption, demodulation, deletion of clauses with trivial equations and – in the case of Horn clauses only – deletion of clauses containing complementary literals. The considered structures are rooted directed labeled graphs, which are general enough to capture most existing equational graph theories, such as those developed for quantum circuits. In contrast to [14], the calculus is able to handle disjunctions as well as interpreted labels, and in contrast to [22], our solution avoids any encoding of graphs into terms, by defining inference rules operating directly on graphs.

From a practical point of view, it would be interesting to get more general redundancy criteria, to reduce the branching factor and improve the efficiency of the procedure. In particular, is it possible to define a version of the calculus in which tautology deletion is allowed, even for non Horn clauses? As evidenced by Example 22, this would require to define a new equational factorization rule, allowing for non trivial superposition inferences within a single clause.

Another interesting issue is to add variables denoting not only labels, but also graphs. This would allow for instance to synthesize graphs satisfying some properties. As graphs can be viewed as functions with multiple inputs and outputs (denoted by the roots) such an addition would yield a second order logic.

Finally, it would be interesting to identify fragments for which the calculus terminates, ensuring decidability of the satisfiability problem. In contrast to terms, the calculus does not terminate (and the satisfiability problem is undecidable) for ground unit clauses [14], hence strong restrictions on the shape of the graphs are required to ensure termination.

References

1. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.

2. L. Bachmair and H. Ganzinger. Strict basic superposition. In C. Kirchner and H. Kirchner, editors, *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, volume 1421 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 1998.
3. M. Backens and A. Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *arXiv preprint arXiv:1805.02175*, 2018.
4. F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski, and F. Zanasi. Confluence of graph rewriting with interfaces. In *26th European Symposium on Programming (21/04/17 - 28/04/17)*, February 2017.
5. F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski, and F. Zanasi. String diagram rewrite theory I: rewriting with frobenius structure. *J. ACM*, 69(2):14:1–14:58, 2022.
6. J. H. Brenas, R. Echahed, and M. Strecker. Verifying graph transformation systems with description logics. In *Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings*, volume 10887 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2018.
7. R. Caferra, R. Echahed, and N. Peltier. A term-graph clausal logic: Completeness and incompleteness results. *Journal of Applied Non-classical Logics*, 18(4):373–411, 2008.
8. C. Chareton, S. Bardin, F. Bobot, V. Perrelle, and B. Valiron. An automated deductive verification framework for circuit-building quantum programs. In N. Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 148–177. Springer, 2021.
9. A. Clément, N. Heurtel, S. Mansfield, S. Perdrix, and B. Valiron. Lo_v-calculus: A graphical language for linear optical quantum circuits. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
10. A. Clément and S. Perdrix. PBS-calculus: A graphical language for quantum-controlled computations. *arXiv preprint arXiv:2002.09387*, 2020.
11. B. Coecke and R. Duncan. Tutorial: Graphical calculus for quantum circuits. In *International Workshop on Reversible Computation*, pages 1–13. Springer, 2012.
12. A. Corradini, D. Duval, R. Echahed, F. Prost, and L. Ribeiro. The PBPO graph transformation approach. *J. Log. Algebraic Methods Program.*, 103:213–231, 2019.
13. R. Echahed. Inductively sequential term-graph rewrite systems. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings*, volume 5214 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2008.
14. R. Echahed, M. Echenim, M. Mhalla, and N. Peltier. A superposition-based calculus for diagrammatic reasoning. In N. Veltri, N. Benton, and S. Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 10:1–10:13. ACM, 2021.

15. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.
16. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages and Tools.* World Scientific, 1999.
17. H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2987 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2004.
18. H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution.* World Scientific, 1999.
19. H. Ehrig, M. Pfender, and H. J. Schneider. Graph-grammars: An algebraic approach. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 167–180, 1973.
20. J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 1–94. World Scientific, 1997.
21. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer-Verlag, 1990.
22. J. Gorard, M. Namuduri, and X. D. Arsiwalla. Zx-calculus and extended wolfram model systems II: fast diagrammatic reasoning with an application to quantum circuit simplification. *CoRR*, abs/2103.15820, 2021.
23. A. Habel and K. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Math. Struct. Comput. Sci.*, 19(2):245–296, 2009.
24. A. Hadzihasanovic. The algebra of entanglement and the geometry of composition. *arXiv preprint arXiv:1709.08086*, 2017.
25. A. Kissinger and V. Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *International Conference on Automated Deduction*, pages 326–336. Springer, 2015.
26. C. M. Poskitt and D. Plump. A hoare calculus for graph programs. In H. Ehrig, A. Rensink, G. Rozenberg, and A. Schürr, editors, *Graph Transformations - 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 - - October 2, 2010. Proceedings*, volume 6372 of *Lecture Notes in Computer Science*, pages 139–154. Springer, 2010.
27. A. Rensink. The GROOVE simulator: A tool for state space generation. In *Second International Workshop on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2003*, volume 3062 of *LNCS*, pages 479–485. Springer, 2003.
28. A. Riazanov and A. Voronkov. Vampire 1.1 (system description). In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 376–380. Springer, 2001.
29. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.
30. S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, higher, stronger: E 2.3. In P. Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*,

volume 11716 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2019.

31. D. Varró. Automated formal verification of visual modeling languages by model checking. *Journal of Software and Systems Modeling*, 3(2):85–113, May 2004.
32. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.