Paramodulation without Duplication

Christopher Lynch
INRIA Lorraine et CRIN
Campus Scientifique
BP 101
54602 Villers-lès-Nancy Cedex
France

Abstract

The resolution (and paramodulation) inference systems are theorem proving procedures for first-order logic (with equality), but they can run exponentially long for subclasses which have polynomial time decision procedures, as in the case of SLD resolution and the Knuth-Bendix completion procedure, both in the ground case. Specialized methods run in polynomial time, but have not been extended to the full first–order case. We show a form of Paramodulation which does not copy literals, which runs in polynomial time for the ground case of the following four subclasses: Horn Clauses with any selection rule, any set of Unit Equalities (this includes Completion), Equational Horn Clauses with a certain selection rule, and Conditional Narrowing.

1 Introduction

Since the early 1960's, automated deduction has been a popular topic of research. Resolution remains the most popular method, along with its extension to equality of Paramodulation [20]. Methods have been developed to restrict the search space of Resolution and Paramodulation proof procedures, and new theorem proving procedures have been developed. Much of this research has been to make the theorem prover more efficient. The main technique of showing the efficiency of a theorem prover is to compare it with another prover on a few test problems, with some recent emphasis to develop a database of test problems.

During this same period of time, the field of Analysis of Algorithms has been developed to compare the running time of different algorithms. The complexity of running times of Automated Theorem Provers is not analyzed, because the problem of theorem proving is undecidable. However, some methods have been developed to solve certain important classes of formulas in polynomial time. One way to show efficiency of a general theorem prover would be to show that it decides these important classes in polynomial time. Up until now, these classes have only been polynomially decided by specialized procedures which have not been extended to more general settings. Resolution and Paramodulation are exponential for these classes. This paper shows how special data structures can be used to make Resolution and Paramodulation run in polynomial time on these classes, while not requiring the inference system to use special strategies.

One class which can be decided in polynomial time is the class of propositional Horn Clause formulas. Goal–directed strategies are useful in theorem proving. In an important recent paper, Plaisted [17, 18] has examined the behavior of many general purpose goal–directed theorem proving methods on this class. His conclusion is that the only strategies which are polynomial are those strategies which use methods to save goals as they are solved and avoid repeatedly solving them. These techniques generally only work for Horn clauses and not for first order logic. Plaisted showed that only the MESON and Model Elimination strategies [9] with unit lemmas and caching [1], the simple and modified problem reduction formats [15, 16] and the Hyper–linking strategy [8] are polynomial. However, the first two of these strategies (with caching) are not complete for first–order logic; the second two are inefficient on non–Horn clauses, and have not been extended to equality. He also notes that none of the resolution strategies are polynomial, and suggests that maybe SLD–resolution should be based on another strategy. We note that caching methods (often called memoing) [22, 23] have been applied to SLD–resolution, but the technique is not complete for non–Horn clauses. Local Simplification [10] is a general strategy for first–order equational theorem proving that always has polynomial size proofs in the case of SLD–resolution of ground clauses, but the search space is exponential.[1]

Knuth–Bendix completion is a method to convert a set of equations to an equivalent canonical set. It corresponds to paramodulation in the case where all clauses are positive unit equalities. Polynomial time algorithms, based on congruence closure, have been given to convert a set of equations without variables into an equivalent canonical set in polynomial time [5, 21], but these algorithms are not Completion and do not work when the equations contain variables. Recently, it has been shown that ground completion with structure sharing is polynomial if a strategy is used which applies critical pairs in a certain order [19].

In this paper, we define a method of performing paramodulation without copying literals. The terms are stored in a hypergraph. To perform resolution or paramodulation a hyperedge is added to the graph. The hyperedge is labelled by the unification problem. Therefore, many inferences are performed simultaneously. The graph can store exponentially many clauses and exponentially large clauses in a polynomial space. It can also store infinitely many clauses in a finite space. In some sense, it can be seen as an extension of the caching technique for logic programming and the congruence closure technique for completion to the general first order case. The Clause Graph Resolution method [7] superficially resembles our method, but it copies literals, so it does not have a polynomial search space for the above cases. The Clause Graph Resolution method was developed to improve the efficiency of processes such as finding literals to unify, and those techniques can be applied equally well here.

The format of this paper is as follows. In section 2, we give some definitions. Then in section 3 we explain how clauses are represented in *Paramodulation without Duplication* (PWD). In section 4, we explain how inferences cause edges to be added to the graph. In section 5, we show the complexity results of this method, all for classes without variables. Specifically we show that the search space of PWD is polynomial for the class of Horn Clauses for any selection rule, including goal–directed selection rules. This differs with caching results mentioned above which are only complete for certain kinds of goal–directed selection rules like the left–to–right strategy of PROLOG. We show the search space is polynomial for the class consisting of only unit equalities and disequalities. This shows that it is polynomial for completion. The search space is also polynomial for conditional narrowing,[2] and for Equational Horn Clauses under certain selection rules.

[1] The existence of polynomial–sized proofs has been studied by Haken[6] and others but Automated Deduction researchers are concerned with the efficiency of searching for a proof.

[2] See [11] for an explanation of this class.

2 Preliminaries

Our setting is equational clauses. We have predicate symbols, function symbols and variables. A term is a variable or an n–ary function symbol applied to n terms. A literal is an n–ary predicate symbol (possibly negated) applied to n terms. The equality symbol $\approx$ is a special binary predicate symbol, represented in infix notation. The negation of $\approx$ is written as $\not\approx$. An *unconstrained clause* is a disjunction of literals, also viewed as a multiset. An object without variables is *ground*. The empty clause is denoted by $\Box$. Let E be the set of equality axioms, i.e., the axioms for reflexivity, symmetricity, transitivity, and substitution of equals for equals. A set of clauses S is *satisfiable* if $S \cup E$ has a model, otherwise S is *unsatisfiable*.

We use the usual definitions of Automated Deduction (see [9]). A *substitution* is a function from the set of variables to the set of terms, that is almost everywhere the identity. A substitution is identified with the homomorphic extension of itself. Composition of substitutions σ and θ is defined so that $t\sigma\theta = (t\sigma)\theta$ for all t. A substitution is a *renaming substitution* if it is a 1–1 function whose codomain is the set of variables. We assume that all the variables that have appeared are renamed to *fresh variables* (variables that have not appeared).

We use the symbol $\doteq$ as a syntactic equality symbol. An equational constraint is a conjunction of syntactic equalities. A solution of an equational constraint is a substitution. Every substitution σ is a solution of $\top$. σ is a solution of $s \doteq t$ if and only if $s\sigma = t\sigma$. σ is a solution of $\varphi_1 \wedge \varphi_2$ if and only if σ is a solution of φ_1 and σ is a solution of φ_2. A *constrained clause* is a pair of an unconstrained clause C and an equational constraint φ represented as $C [\![\varphi]\!]$. The ground clause $C\sigma$ is an *instance* of $C [\![\varphi]\!]$ if σ is a solution of φ. The clause $C_1 [\![\varphi_1]\!]$ is a *variant* of $C_2 [\![\varphi_2]\!]$ if $C_1 [\![\varphi_1]\!]$ and $C_2 [\![\varphi_2]\!]$ have the same instances.

Given a clause C, a *selection rule* is a function from $C [\![\theta]\!]$ to a set of literals $\Gamma \subseteq C$. If A is in Γ, we say that A is *selected* in $C [\![\theta]\!]$. We assume a well–founded reduction ordering $<$, total on ground terms, identified with its multiset extension. We call a literal A *maximal* in $C [\![\theta]\!]$ if $A \in C$ and there is a solution σ of θ such that $A\sigma \geq B\sigma$ for all $B \in C$.

We now present the Basic Paramodulation inference rules [4, 13]. They differ from the standard Paramodulation inference rules in that the most general unifier is not applied to the conclusion of the inference. Instead, it is saved as an equational constraint in the conclusion of the inference. This is more restrictive than the standard inference rules, because disallowing paramodulation inferences into variables corresponds to never allowing a paramodulation inference into a substitution position. A clause may be removed when its equational constraint is not satisfied.

Our presentation of Basic Paramodulation differs from the standard presentation, since we explicitly give the renaming substitution that must be applied to at least one of the premises of the inference. Normally, it is just assumed that the clauses are renamed. However, since the handling of the renaming is very important in PWD, we explicitly state it here to clarify its use.

Resolution

$$\frac{\Gamma \vee A\,[\![\,\theta_1\,]\!] \qquad \neg A' \vee \Delta\,[\![\,\theta_2\,]\!]}{\Gamma\eta \vee \Delta\,[\![\,A\eta \doteq A' \wedge \theta_1\eta \wedge \theta_2\,]\!]}$$

where A is selected in $\Gamma \vee A\,[\![\,\theta_1\,]\!]$, $\neg A'$ is selected in $\neg A' \vee \Delta\,[\![\,\theta_2\,]\!]$, and η is a new renaming substitution. Note that η is only applied to the left premise.

Positive Factoring

$$\frac{\Gamma \vee A \vee A'\,[\![\,\theta\,]\!]}{\Gamma \vee A\,[\![\,A \doteq A' \wedge \theta\,]\!]}$$

where A' is selected in $\Gamma \vee A \vee A'\,[\![\,\theta\,]\!]$.

Basic Paramodulation

$$\frac{\Gamma \vee s \approx t\,[\![\,\theta_1\,]\!] \qquad L[s'] \vee \Delta\,[\![\,\theta_2\,]\!]}{\Gamma\eta \vee L[t\eta] \vee \Delta\,[\![\,s\eta \doteq s' \wedge \theta_1\eta \wedge \theta_2\,]\!]}$$

where s' is not a variable, $s \approx t$ is selected in $\Gamma \vee s \approx t\,[\![\,\theta_1\,]\!]$, $L[s']$ is selected in $L[s'] \vee \Delta\,[\![\,\theta_2\,]\!]$, $s \not\leq t$, if $L[s']$ is a disequation then s' appears on a maximal side of the disequation, if $L[s']$ is of the form $u[s'] \approx v$ then $u[s']$ is maximal in $\{u[s'], v\}$ or some ground instance $v\sigma$ of v is identical with a ground instance $w\sigma$ of one side of an equation in Δ[3], and η is a new renaming substitution.

Equation Resolution

$$\frac{s \not\approx t \vee \Gamma\,[\![\,\theta\,]\!]}{\Gamma\,[\![\,s \doteq t \wedge \theta\,]\!]}$$

where $s \not\approx t$ is selected in $s \not\approx t \vee \Gamma\,[\![\,\theta\,]\!]$.

Let Sel^{max} be a selection rule that selects all maximal literals in each clause. Let Sel^- be a selection rule that selects a negative literal in each clause containing one, otherwise selects all maximal literals. Let Sel^+ be a selection rule which selects all positive literals in each clause containing one, otherwise selects a negative literal. Let Sel be a selection rule which selects one literal in each clause. Then BP^{max} is the set of the above inference rules with selection rule Sel^{max}, BP^- the above set with Sel^-, BP^+ the above set with Sel^+, and BP the above set with Sel. Note that the completeness of BP on a class implies the completeness of the more specific inferences systems. Similarly, any class where BP has a polynomial search space implies a polynomial search space for the other inference systems.

We can define a notion of *redundant* [3], so that a clause may be removed when it is redundant. This is a general notion that implies practical deletion methods like demodulation and subsumption. Here we state a property that is necessary to hold for redundancy to preserve completeness. If $C\,[\![\,\theta\,]\!]$ is a clause and S is a set of clauses, then $C\,[\![\,\theta\,]\!]$ is *redundant in* S if for every ground instance D of $C\,[\![\,\theta\,]\!]$ there exist ground instances $C_1, \cdots, C_n$ of S such that $C_1 \wedge \cdots \wedge C_n$ logically implies D and for all i, $C_i < D$ and C_i is *reduced relative to* D.[4] In practice, we use whichever forms of redundancy are useful. For example, in some instances of the above inferences, the addition of the conclusion of the inference implies that one of the premises of the inference is redundant. These cases of redundancy are easy to detect and quite useful to perform in practice.

A *theorem proving derivation* models an automated theorem prover [2]. It is used to prove soundness and completeness results, but we have extended it to refer to complexity. If I is an inference system and S is a set of clauses, then an *I-inference from S* is an instance of one of the inferences in I, where the premises are clauses in S. A sequence $S_0, S_1, S_2, \cdots$ of sets of clauses is called an *I–theorem proving derivation* from S_0 if every S_{i+1} is obtained by adding the conclusion of an I-inference from S_i or deleting a redundant clause in S_i. No inference is performed twice in the sequence. Furthermore, every I–inference applied to clauses in $S_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} S_j$ is redundant in S_∞. If the sequence is of the form $S_0, S_1, S_2, \cdots, S_n$ we say that the *length* of the sequence is n. Otherwise we say that the *length* of the sequence is ∞. A *class* is defined to be a set of sets of clauses. For a class $\mathcal{C}$ and a function f, we say that the *length of I over* $\mathcal{C}$ *is bounded by f* if for all $S_0 \in \mathcal{C}$, the longest I–theorem proving derivation from S_0 has length less than or equal to $f(n)$ where n is the number of symbols in S_0. An inference procedure I *decides* a class $\mathcal{C}$ if the length of I over

[3]This is called Merging Paramodulation in [3]. We prefer it to Equational Factoring for this formalism.

[4]See [4] for a definition of *reduced relative to*.

$\mathcal{C}$ is bounded by some function f and there is some recursive procedure to calculate each S_{i+1} from S_i for each such sequence. An inference procedure I *polynomially decides* $\mathcal{C}$ if the length of I over $\mathcal{C}$ is bounded by a polynomial f and S_{i+1} can be calculated from S_i on each such sequence in polynomial time.

An inference system I is *sound* if for any I–theorem proving derivation from a satisfiable S_0, $\Box \notin S_\infty$. I is *complete* if for any I–theorem proving derivation from an unsatisfiable S_0, $\Box \in S_\infty$.

Let $\mathcal{HC}$ be the class of all sets of ground horn clauses without equations. Let $\mathcal{HCE}$ be the class of all sets of ground horn clauses with equations. Let $\mathcal{UE}$ be the class of all sets of ground equational clauses with just one literal per clause. Let $\mathcal{CN}$[5] be the class of all sets of ground horn clauses such that every Paramodulation inference among positive literals of any two clauses is redundant.

Theorem 1 *The inference rules BP^{max} [4, 13] and BP^- [4] are sound and complete. BP [11] (and therefore also BP^+) is sound but only complete for $\mathcal{HCE}$ and its subclasses.*[6] *None of the given inference systems polynomially decides $\mathcal{HCE}$, $\mathcal{UE}$ or $\mathcal{CN}$. Only BP^- polynomially decides $\mathcal{HC}$ (see [17, 18] for more details).*

The classes which cannot be polynomially decided by these general inference systems do have specialized inference procedures which polynomially decide them. The inference system BP^+ is especially important, because it is a goal–directed inference system for $\mathcal{HC}$ and $\mathcal{CN}$. BP^+ on $\mathcal{HC}$ is called SLD–resolution. In this paper we show how clauses can be represented so that BP polynomially decides $\mathcal{HC}$ and $\mathcal{UE}$ and so that BP^- polynomially decides $\mathcal{HCE}$ and BP^+ polynomially decides $\mathcal{CN}$.

3 Clause Representation

Now we show how to represent the set of clauses as a directed hypergraph G, so that literals and terms are not copied when inferences are performed. Each clause in the initial set of clauses is represented as a tree (or dag) in the usual way. The result of an inference will be to add a new hyperedge to the graph, labelled by an equational constraint and a renaming (as in Basic Paramodulation). This will make it easy to read off the clauses from the hypergraph.

The vertices of the graph are labelled with symbols from the language. Each vertex is labelled with a variable, constant, function symbol, predicate symbol, negated predicate symbol, or the symbol $\vee$.

All edges in the graph are directed. There are two kinds of edges: *subterm edges* and *replacement edges*. Each subterm edge is labelled by an integer. If the source of a subterm edge is labelled by an n–ary symbol, then it will have n subterm edges directed out of it, labelled by each of the integers from 1 to n, indicating the index of the subterm. The subterm edges are used to represent the set of clauses that we want to perform inferences on. For example the set of clauses $P(a,b) \vee f(c,d) \approx e$ is represented by the following graph.

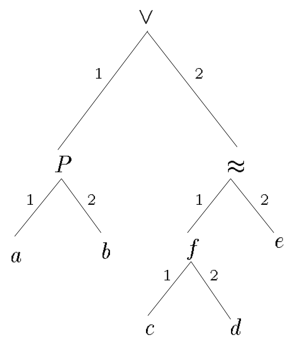

Figure 1: Initial set of clauses

Note in this example that the $\vee$ symbol is assumed to be binary for this clause, although it may appear elsewhere in the graph with a different arity. The arity for $\approx$ is always 2.

The replacement edges are added to the graph when inferences are performed. Each replacement edge is labelled with a renaming substitution and a path. A *path* is a sequence of edges $< e_1, \cdots, e_n >$ so that for each $i < n$, the target of e_i is the source of e_{i+1}. We use Π for paths and the comma is overloaded, so that $< e, \Pi >$ refers to the path Π with e added to the beginning and $< \Pi_1, \Pi_2 >$ indicates Π_1 concatenated with Π_2. The path is only necessary for Paramodulation. It is not necessary for Resolution or Completion. The source of a replacement edge is labelled by a function symbol, constant or (usually negated) predicate symbol. Replacement edges are of two types: *rewrite edges* and *resolution edges*. The target of a rewrite edge is always labelled by a function symbol, constant or variable. The target of a resolution edge is labelled by a (possibly negated) predicate symbol.

Subterm edges are grouped together as subterm hyperedges. A subterm hyperedge is a set of subterm edges with the same source, such that if the source is

[5] $\mathcal{CN}$ stands for conditional narrowing.

[6] In [11], the substitutions are required to be applied to clauses with positive literals, and examples are given which show that the definition of redundancy must be modified.

labelled with an n–ary symbol then there are n edges in the set labelled with the integers from 1 to n. In an initial graph, every edge has one subterm hyperedge leading from it, except the nodes labelled by constants, variables, and 0–ary predicate symbols (see Figure 1).

Replacement edges are also grouped together as replacement hyperedges. A replacement hyperedge is a set of replacement edges with the same source. A replacement hyperedge will contain at most one rewrite edge. Each replacement hyperedge is labelled with an equational constraint. When a critical pair inference is performed, a replacement hyperedge containing just a rewrite edge is added to the graph. When a Resolution Inference is performed, a replacement hyperedge is added which contains only resolution edges. When a paramodulation inference is performed, a replacement hyperedge is added containing one rewrite edge and zero or more resolution edges. Replacement hyperedges are labelled with the equational constraint corresponding to the inference. Each edge in a replacement hyperedge is labelled with a renaming substitution to be applied to the target term. We give some examples here and leave the formal definition for the next section. In all examples, any omitted labellings are assumed to be trivial (i.e., missing equational constraints are assumed to be $\top$, missing renaming substitutions are id, and a missing path is some non–existing path). Similarly, clauses written as unconstrained clauses are assumed to have a constraint of $\top$. First consider the following Resolution inference:

$$\frac{p \vee q \vee r \qquad \neg r \vee s \vee t}{p \vee q \vee s \vee t}$$

The conclusion of the inference is the same as the right premise, except that the literal $\neg r$ is replaced by the literals p and q, so we draw a replacement hyperedge containing a resolution edge from the vertex labelled by $\neg r$ to the vertex labelled by p and from the vertex labelled by $\neg r$ to the vertex labelled by q. Suppose we also perform the inference

$$\frac{\neg r \vee s \vee t \qquad \neg t \vee p'}{\neg r \vee s \vee p'}$$

or the inference

$$\frac{p \vee q \vee s \vee t \qquad \neg t \vee p'}{p \vee q \vee s \vee p'}$$

We represent these inferences in Figure 2.

Consider the following Critical Pair inference:

$$\frac{f(f(x)) \approx f(g(x)) \qquad f(f(x)) \approx f(g(x))}{f(f(g(x))\rho) \approx f(g(x)) \llbracket f(f(x))\rho \doteq f(x) \rrbracket}$$

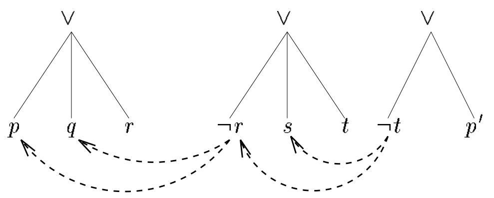

Figure 2: Resolution Inference

The subterm $f(x)$ in the second premise is unified with $f(f(x))\rho$ and replaced by $f(g(x))\rho$ from the first premise. We must label the new rewrite edge with the equational constraint $f(f(x))\rho = f(x)$ and the renaming substitution ρ. It is represented graphically in figure 3.

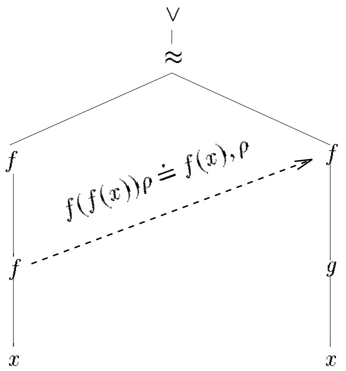

Figure 3: Critical Pair Inference

We define an unfolding of the graph G to be a tree T such that each vertex and hyperedge in T is also in G, although vertices and hyperedges may be duplicated in T. If there is an edge e from u to v in T, then e is directed from u to v in G. Each node has only one hyperedge leading from it in T. Also, we require that each leaf of T is labelled by a variable, constant, or 0–ary predicate symbol.

We will define $\hat{T}$ to be the element that a tree T represents. If the root of an unfolding T is labelled by $\vee$, a predicate symbol or a negated predicate symbol, then $\hat{T}$ is a constrained clause. If the root of T is a variable, constant or function symbol, then $\hat{T}$ is a constrained term–clause pair $(t, C) \llbracket \varphi \rrbracket$. If v is a vertex, and U is the set of all unfoldings rooted at v, then $\hat{v} = \bigcup_{T \in U} \hat{T}$. Since the graph may have cycles, $\hat{v}$ can be infinite, although each element of $\hat{v}$ is finite.

Let T be a tree whose root is labelled with $\vee$. We say that v *represents* $\hat{T}$ if v is the root of T. If T' is a subtree of T such that $\hat{T'} = L \vee C ⟦ \varphi ⟧$ and there is no subtree T'' of T' such that $\hat{T''} = L \vee C' ⟦ \varphi' ⟧$ for some C' and φ', then we say that v *represents* L in $\hat{T}$ if v is the root of T'. If T' is a subtree of T such that $\hat{T'} = (t, C) ⟦ \varphi ⟧$ and there is no subtree T'' of T' such that $\hat{T''} = (t, C') ⟦ \varphi' ⟧$ for some C' and φ', then we say that v *represents* t in $\hat{T}$ if v is the root of T'.

Before defining $\hat{T}$, we must give a few definitions concerning trees and graphs. A tree T can be represented as the set $\{u, (e_1, T_1), \cdots (e_n, T_n)\}$, where u is the root of T, $T_1, \cdots, T_n$ are subtrees of T and for each i, e_i is the edge in T from u to the root of T_i. If $n = 0$, then T is the trivial tree.

Let T be a tree of the form $\{u, (e_1, T_1), \cdots (e_n, T_n)\}$ and Π be a path. We inductively define the notion of *prefix* and the notation T_Π meaning the subtree of T rooted at the path Π. If Π is the empty path, then Π is a prefix of T and $T_\Pi = T$. Otherwise, Π is a prefix of T if Π is the sequence of edges $< e_i, \Pi' >$ for some i and Π' is a prefix of T_i. In this case $T_\Pi = T_{i\Pi'}$. If Π is not a prefix of T, then T_Π is the empty tree. If Π is a prefix of T and u is the source of some edge in Π and v is the target of the final edge in Π, then u is *above* v and v is *below* u.

We define the removal of a path from a tree as follows. If T is a tree and Π is a path, then $T - \Pi$ is the tree T with T_Π replaced by the empty tree. If T_Π is the empty tree, then $T - \Pi = T$. We also define an ordering on trees such that $T < T'$ if the number of replacement edges in T is smaller than the number of replacement edges in T'.

Each $\hat{T}$ can be defined inductively. If T is the empty tree, then $\hat{T} = \Box ⟦ \top ⟧$. If T has a single node labelled by a 0–ary predicate symbol P, then $\hat{T} = P ⟦ \top ⟧$. Negated predicate symbols are treated similarly. If T has a single node labelled by t where t is a variable or a constant symbol then $\hat{T} = (t, \Box) ⟦ \top ⟧$.

If T is of the form $\{u, (e_1, T_1), \cdots, (e_n, T_n)\}$ where $\{e_1, \cdots, e_n\}$ is a subterm hyperedge then each e_i is labelled with i. If u is labelled by $\vee$ and $\hat{T_i} = C_i ⟦ \varphi_i ⟧$ for each i, then $\hat{T} = C_1 \vee \cdots \vee C_n ⟦ \varphi_1 \wedge \cdots \wedge \varphi_n ⟧$. If u is labelled by a predicate symbol P and $\hat{T_i} = (t_i, C_i) ⟦ \varphi_i ⟧$ for each i, then $\hat{T} = P(t_1, \cdots, t_n) \vee C_1 \vee \cdots \vee C_n ⟦ \varphi_1 \wedge \cdots \wedge \varphi_n ⟧$. Negated predicate symbols are treated similarly. If u is labelled by a function symbol f and $\hat{T_i} = (t_i, C_i) ⟦ \varphi_i ⟧$ for each i, then $\hat{T} = (f(t_1, \cdots, t_n), C_1 \vee \cdots \vee C_n) ⟦ \varphi_1 \wedge \cdots \wedge \varphi_n ⟧$.

Suppose $T = \{u, (e_1, T_1), \cdots, (e_n, T_n)\}$ where u is labelled with a (usually negated) predicate symbol, $e = \{e_1, \cdots, e_n\}$ is a replacement hyperedge, and the label of each e_i is (ρ_i, Π_i), where ρ_i is a renaming substitution and Π_i is a path. The hyperedge e is labelled by an equational constraint φ. Suppose that for each i, $\hat{T_i}' = C_i ⟦ \varphi_i ⟧$ where $T_i' = T_i - \Pi_i$. Then $\hat{T} = C_1\rho_1 \vee \cdots \vee C_n\rho_n ⟦ \varphi \wedge \varphi_1\rho_1 \wedge \cdots \wedge \varphi_n\rho_n ⟧$

In Figure 2, suppose u_p is the node labelled by p, $u_{\neg r}$ is the node labelled by $\neg r$, $u_{\neg t}$ is the node labelled by $\neg t$ and $u_\vee$ is the node labelled by the $\vee$ on the right. Then $\hat{u_p} = \{p\}$, $\hat{u_{\neg r}} = \{\neg r, p \vee q\}$, $\hat{u_{\neg t}} = \{\neg t, \neg r \vee s, p \vee q \vee s\}$ and $\hat{u_\vee} = \{\neg t \vee p', \neg r \vee s \vee p', p \vee q \vee s \vee p'\}$. So u_p represents p in $p \vee q \vee r$ and in $p \vee q \vee s \vee p'$. Also, $u_{\neg r}$ represents $\neg r$ in $\neg r \vee s \vee p'$ and $u_{\neg t}$ represents $\neg t$ in $\neg t \vee p'$.

Suppose $T = \{u, (e', T'), (e_1, T_1), \cdots (e_n, T_n)\}$ where u is labelled with a constant or function symbol, $e = \{e', e_1, \cdots, e_n\}$ is a replacement hyperedge, e' is a rewrite edge, the label of each e_i is (ρ_i, Π_i), and the label of e' is (ρ', Π'). The hyperedge e is labelled by an equational constraint φ. Suppose that for each i, $\hat{T_i}' = C_i ⟦ \varphi_i ⟧$ where $T_i' = T_i - \Pi_i$ and $\hat{T''} = (t', C') ⟦ \varphi' ⟧$. where $T'' = T' - \Pi'$. Then $\hat{T} = (t'\rho', C'\rho' \vee C_1\rho_1 \vee \cdots \vee C_n\rho_n) ⟦ \varphi \wedge \varphi'\rho' \wedge \varphi_1\rho_1 \wedge \cdots \wedge \varphi_n\rho_n ⟧$.

In Figure 3, suppose u_1 is the node labelled by f on the top of the right hand side, u_2 is the node labelled by f on the middle of the left hand side and u_3 is the node labelled by $\approx$. Then $\hat{u_1} = \{(f(g(x)), \Box)\}$, $\hat{u_2} = \{(f(x), \Box), (f(g(x))\rho, \Box) ⟦ f(f(x))\rho \doteq f(x) ⟧\}$, and $\hat{u_3} = \{f(f(x)) \approx f(g(x)), f(f(g(x))\rho) \approx f(g(x)) ⟦ f(f(x))\rho \doteq f(x) ⟧\}$. So u_1 represents $f(g(x))$ in $f(f(x)) \approx f(g(x))$ and in $f(f(g(x))\rho) \approx f(g(x)) ⟦ f(f(x))\rho \doteq f(x) ⟧$. Also, u_2 represents $f(x)$ in $f(f(x)) \approx f(g(x))$, but does not represent $f(g(x))\rho$ in $f(f(g(x))\rho) \approx f(g(x)) ⟦ f(f(x))\rho \doteq f(x) ⟧$.

To understand the meaning of path labels, we refer the reader to figure 4. To simplify the example, the edges are referred to by the labels of the vertices on either end. Let u_Q be the node labelled by Q, u_a be the node labelled by a, $u_{\neg P}$ be the node labelled by $\neg P$, and $u_{\neg Q}$ be the node labelled by $\neg Q$. There are four replacement hyperedges in the graph. One contains a resolution edge from u_a to u_Q and a rewrite edge from u_a to the node labelled by b. One contains a resolution edge from $u_{\neg P}$ to u_Q. One contains a resolution edge from $u_{\neg Q}$ to $u_{\neg P}$ and is labelled with the path $< (\neg P, a)(a, Q) >$. The other one also contains a resolution edge from $u_{\neg Q}$ to $u_{\neg P}$ but it is labelled with path $< (\neg P, Q) >$. Then $\hat{u_Q} = \{Q\}$, $\hat{u_a} = \{(a, \Box), (b, Q)\}$, $\hat{u_{\neg P}} = \{\neg P(a), \neg P(b) \vee Q\}$, and $\hat{u_{\neg Q}} = \{\neg Q, \neg P(a), \neg P(b), \Box\}$. For the meaning of $u_{\neg Q}$, we have thrown away the information at the end of the path (i.e., Q is not included in the clause with $\neg P(b)$ or with $\Box$).

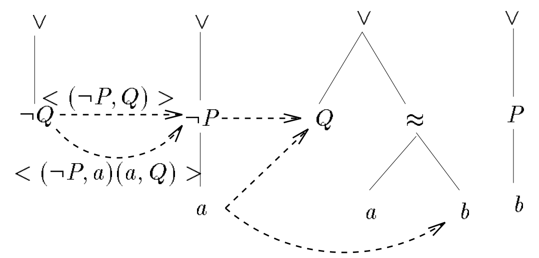

Figure 4: Paramodulation Inference

4 Inference Rules

In the previous section, we have shown how to create a graph from an initial set of clauses, using only subterm edges. We have also defined the syntax and semantics of the graph. We mentioned that the inference rules add replacement edges to the graph. In this section, we define exactly how the inference system adds these edges to the graph.

The clauses that exist in the graph at any moment of a theorem proving derivation are represented by the set $\bigcup_{v \in U} \hat{v}$ where U is the set of all vertices in the graph labelled with $\vee$. At each step of the inference procedure, the theorem prover must decide what edge to add next. To do that, it must decide what Basic Paramodulation inference must be performed. It is possible to do this without calculating all the clauses. It is only necessary to know which literals in the graph are selected in some clause. We must also know which clauses contain a selected positive literal. For negative literals we only need to know if they are selected in some clause, but not which clause they are selected in. In this section we describe what edge the theorem prover must add once it decides the inference to perform. In many cases, the theorem prover will not perform the necessary inference. Instead, it will add an edge corresponding to another inference. But because of the addition of that edge, the required inference will also be performed.

The theorem prover must halt when no inferences can be performed or the empty clause is found. One simple way to search for the empty clause, is by using the inductive definition of resolution edges to calculate a fixed point of $\hat{v}$ for each vertex v. To begin with, $\hat{v} = \emptyset$ for all v. Then at each stage, we apply a single application of the inductive definition of replacement hyperedges. If that places $\Box [\![\varphi]\!]$ in $\hat{v}$ for some φ, then we add $\Box [\![\varphi]\!]$ to $\hat{v}$. Otherwise, we don't add anything to $\hat{v}$. Eventually, if a tree rooted at a vertex labelled with $\vee$ represents the empty clause with a satisfiable constraint, then the empty clause is present. Each step of this process can be done in polynomial time in the size of the constraints. For the ground case, the procedure can not take more steps than the number of vertices in the graph, because it can only add $\Box$ once to each vertex. If $\Box$ is added to no vertices at some step, then the fixed point is reached, and the search for the empty clause is terminated. For the non–ground case, the procedure may proceed infinitely long, so it may be necessary to dovetail the search for the empty clause with the addition of new edges which represent inferences.

Now we explain how to perform each inference. First, consider the resolution inference rule.

$$\frac{\Gamma \vee A [\![\theta_1]\!] \qquad \neg A' \vee \Delta [\![\theta_2]\!]}{\Gamma\eta \vee \Delta [\![A\eta \doteq A' \wedge \theta_1\eta \wedge \theta_2]\!]}$$

Let T_1 be the tree which represents the clause $\Gamma \vee A [\![\theta_1]\!]$, and T_2 the tree which represents the clause $\neg A' \vee \Delta [\![\theta_2]\!]$. Let u_1 be the vertex that represents A in T_1. Let u_2 be the vertex which represents $\neg A'$ in T_2. Let $T_1{}'$ be the smallest subtree of T_1 which has the same root as T_1 and for which u_1 still represents A. Then $\hat{T_1}'$ is a clause of the form $\Gamma' \vee A [\![\theta_1{}']\!]$. Let $T_2{}'$ be the smallest subtree of T_2 which has u_2 as a root and u_2 still represents $\neg A'$. Then $\hat{T_2}'$ is a clause of the form $\neg A' \vee \Delta' [\![\theta_2{}']\!]$. Let U_1 be the multiset of vertices which represent one of the literals of Γ' in $\hat{T_1}'$. Each u in U_1 is the root of some tree T_u such that $\hat{T_u} = L$ for some literal L in Γ'. Let ρ_u be the renaming applied to L in $\hat{T_1}'$. Let U_2 be the multiset of vertices which represent one of the literals of Δ' in $\hat{T_2}'$. Each u in U_2 is the root of some tree T_u such that $\hat{T_u} = L$ for some literal L in Δ'. Let ρ_u be the renaming applied to L in $\hat{T_2}'$. The resolution we will directly perform is the following

$$\frac{\Gamma' \vee A [\![\theta_1{}']\!] \qquad \neg A' \vee \Delta' [\![\theta_2{}']\!]}{\Gamma'\eta \vee \Delta' [\![A\eta \doteq A' \wedge \theta_1{}'\eta \wedge \theta_2{}']\!]}$$

The clause $\neg A' \vee \Delta' [\![\theta_2{}']\!]$ is not actually a clause. It is only a part of a clause. The inference is performed on all clauses that contain $\neg A' \vee \Delta' [\![\theta_2{}']\!]$.

To perform the resolution, we add a replacement hyperedge e labelled by the equational constraint $A\eta \doteq A' \wedge \theta_1{}'\eta \wedge \theta_2{}'$. If u is a vertex in U_2 then e will contain a resolution edge from u_2 to u labelled with ρ_u. If u is a vertex in U_1 then e will contain a resolution edge from u_2 to u labelled with $\rho_u\eta$. If u is a vertex in U_1 above u_1 in $T_1{}'$, then there is some path Π from u to u_1 in $T_1{}'$. So the edge from u_2 to u will be labelled by the path Π.

Additionally, we may need to modify the path information on some existing paths. If u is a vertex in U_2 below u_2 such that an edge e_u is added from u_2 to u. let Π_u be the path from u_2 to u. Then, for every edge e' of the form $< \Pi_1, \Pi_u, \Pi_2 >$, from some vertex v_1 to some vertex v_2, a new edge is added from v_1 to v_2 which has the same label as e' except that path information is $< \Pi_1, e_u, \Pi_2 >$ instead of $< \Pi_1, \Pi_u, \Pi_2 >$.

When the new replacement hyperedge is added to the graph, several resolution inferences are performed. Let T_3 be the tree formed from T_2 by removing the hyperedge from u_2 in T_2 and adding e and T_1, then removing all the edges and vertices no longer reachable from the root of T_3. Then T_3 represents $\Gamma\eta' \vee \Delta [\![A\eta' \doteq A' \wedge \theta_1\eta' \wedge \theta_2]\!]$, where $\eta' = \rho\eta$ and ρ is the renaming applied to $\neg A'$. The only difference between this and the inference we wanted to perform is that the renaming may be different.

We give two illustrations of the resolution rule. First consider Figure 2. The reader may verify that the hyperedge on the left is created from a resolution inference among $p \vee q \vee r$ and $\neg r \vee s \vee t$. The hyperedge on the right is created by an inference among the clauses $p \vee q \vee s \vee t$ and $\neg t \vee p'$. According to the above notation, T_1 is the tree representing $p \vee q \vee s \vee t$ and T_2 represents $\neg t \vee p'$. Then ${T_1}'$ is the subtree of T_1 representing $\neg r \vee s \vee t$ and ${T_2}'$ is the subtree of T_2 representing $\neg t$. Therefore we must add a hyperedge, consisting of an edge from the vertex labelled by $\neg t$ to the vertex labelled by $\neg r$ and an edge from the vertex labelled by $\neg t$ to the vertex labelled by s. Note that this also performs the inference we originally wanted to perform.

For another example, consider Figure 4. As mentioned earlier, there are four hyperedges in the graph. The hyperedge directed from the vertex labelled by a is from a paramodulation inference, and we will discuss that below. Suppose we want to perform the following resolution inference:

$$\frac{\neg P(b) \vee Q \qquad \neg Q}{\neg P(b)}$$

Using the above notation, T_1 is the tree rooted at the vertex labelled by the second from the left $\vee$ symbol, containing the hyperedge directed from the vertex labelled a. ${T_1}'$ is the same tree. u_1 is the node labelled Q. u_2 is the node labelled $\neg Q$. The only element of U_1 is the node u labelled by $\neg P$. To perform the resolution inference, we must add an edge from u_2 to u. But since u is above u_1, we must label the edge with the path $< (\neg P, a)(a, Q) >$ from u to u_1. As we mentioned earlier, without the path information, the conclusion to the inference would be $\neg P(b) \vee Q$, which is not correct.

After this inference, we can resolve $\neg P(b)$ and $P(b)$. So u_1 is the vertex labelled by P and u_2 is the vertex labelled by $\neg P$. Let u be the vertex labelled by Q. We need to add a resolution edge from u_2 to u. But since an edge exists between u_2 and u_1 labelled by the path from u_2 to u, we must add another edge from u_2 to u_1 labelled by the edge from u_2 to u.

The other inference rules are similar. Consider positive factoring.

$$\frac{\Gamma \vee A \vee A' [\![\theta]\!]}{\Gamma \vee A [\![A \doteq A' \wedge \theta]\!]}$$

Let T represent the clause $\Gamma \vee A \vee A' [\![\theta]\!]$. Let u_1 be the vertex that represents A in T. Let u_2 be the vertex which represents A' in T. Let ${T_1}'$ be the smallest subtree of T which has u_1 as a root and u_1 represents A. Then $\hat{T_1}'$ is a clause of the form ${\Gamma_1}' \vee A [\![{\theta_1}']\!]$. Let ${T_2}'$ be the smallest subtree of T which has u_2 as a root and u_2 represents A'. Then $\hat{T_2}'$ is a clause of the form ${\Gamma_2}' \vee A' [\![{\theta_2}']\!]$.

Let U_1 be the multiset of vertices which represent one of the literals of ${\Gamma_1}'$ in $\hat{T_1}'$. Each u in U_1 is the root of some tree T_u such that $\hat{T_u} = L$ for some literal L in ${\Gamma_1}'$. Let ρ_u be the renaming applied to L in $\hat{T_1}'$. Let U_2 be the multiset of vertices which represent one of the literals of ${\Gamma_2}'$ in $\hat{T_2}'$. Each u in U_2 is the root of some tree T_u such that $\hat{T_u} = L$ for some literal L in ${\Gamma_2}'$. Let ρ_u be the renaming applied to L in $\hat{T_2}'$.

To perform the positive factoring, we add a replacement hyperedge e labelled by the equational constraint $A \doteq A' \wedge {\theta_1}' \wedge {\theta_2}'$. If u is a vertex in $U_1 \cup U_2$ then e will contain a resolution edge from u_2 to u labelled with ρ_u. Also, e will contain a resolution edge from u_2 to u_1. As in resolution, this may cause path information on other edges to be modified.

Consider the paramodulation inference rule.

$$\frac{\Gamma \vee s \approx t [\![\theta_1]\!] \qquad L[s'] \vee \Delta [\![\theta_2]\!]}{\Gamma\eta \vee L[t] \vee \Delta [\![s\eta \doteq s' \wedge \theta_1\eta \wedge \theta_2]\!]}$$

Let T_1 be the tree which represents the clause $\Gamma \vee s \approx t [\![\theta_1]\!]$, and T_2 the tree which represents the clause $L[s'] \vee \Delta [\![\theta_2]\!]$. Let u_0 be the vertex that represents s, u_1 the vertex that represents t and u_3 the vertex that represents $s \approx t$ in T_1. Let u_2 be the vertex which represents s' in T_2. Let ${T_1}'$ be the smallest subtree of T_1 which has the same root as T_1 and for which u_3 represents $s \approx t'$ for some t' and u_0 represents s in $s \approx t'$. Then $\hat{T_1}'$ is a clause of the form $\Gamma' \vee s \approx t' [\![{\theta_1}']\!]$. Let ${T_2}'$ be the smallest subtree of T_2 which has u_2 as a root and u_2 represents s'. Then $\hat{T_2}'$ is a clause of the form $L'[s'] \vee \Delta' [\![\theta_2']\!]$. Let U_1 be the multiset of vertices which represent one of the literals of Γ' in $\hat{T_1}'$. Each u in U_1 is the root of some tree T_u such that $\hat{T_u} = L_1$ for some literal L_1 in Γ'. Let ρ_u be the renaming applied to L_1 in $\hat{T_1}'$. Let U_2 be the multiset of vertices which represent one of the literals of Δ' in $\hat{T_2}'$. Each u in U_2 is the root of some tree T_u such that $\hat{T_u} = L_2$ for some literal L_2 in Δ'. Let ρ_u be the renaming applied to L_2 in $\hat{T_2}'$.

To perform the paramodulation, we add a replacement hyperedge e labelled by the equational constraint $s\eta \doteq s' \wedge \theta_1'\eta \wedge \theta_2'$. The hyperedge e will contain an edge from u_2 to u_1, labelled with $\rho\eta$ where ρ is the renaming applied to t' in $\hat{T_1}'$. If u is a vertex in U_2 then e will contain an edge from u_2 to u labelled with ρ_u. If u is a vertex in U_1 then e will contain an edge from u_2 to u labelled with $\rho_u\eta$. If u is a vertex in U_1 above u_3 in T_1', then there is some path Π from u to u_3 in T_1'. So the edge from u_2 to u will also be labelled by the path Π. Path information may need to be modified as in resolution.

For an example of a paramodulation inference, consider Figure 4. In that example a paramodulation is performed among the clauses $Q \vee a \approx b$ and $\neg P(a)$. A replacement hyperedge is added which is comprised of a resolution edge from the vertex labelled by a to the vertex labelled by Q, and a rewrite edge from the vertex labelled by a to the vertex labelled by b.

Another example of paramodulation is in figure 3. Here, T_1 and T_2 are the same tree, s' represents $f(x)$, s represents $f(f(x))$ and t represents $f(g(x))$. To perform the paramodulation we add the rewrite edge in the graph. We called the renaming ρ, so that is the label of the rewrite edge, and the replacement hyperedge is labelled with the constraint $f(f(x))\rho \doteq f(x)$.

Finally, we consider Equation Resolution.

$$\frac{s \not\approx t \vee \Gamma [\![\theta]\!]}{\Gamma [\![s \doteq t \wedge \theta]\!]}$$

Let T be the tree which represents the clause $s \not\approx t \vee \Gamma [\![\theta]\!]$. Let u' be the vertex that represents $s \not\approx t$ in T. Let T' be the smallest subtree of T which is rooted at u' and u' represents $s \not\approx t$. Then T' represents a clause of the form $s \not\approx t \vee \Gamma' [\![\theta']\!]$.

Let U be the multiset of vertices which represent one of the literals of Γ' in $\hat{T}'$. Each u in U is the root of some tree T_u such that $\hat{T_u} = L$ for some literal L in Γ'. Let ρ_u be the renaming applied to L in $\hat{T}'$.

To perform the equation resolution, we add a replacement hyperedge e labelled by the equational constraint $s \doteq t \wedge \theta'$. If u is a vertex in U then e will contain an edge from u' to u labelled with ρ_u. The path information may need to be modified as in resolution.

Redundant clauses may be removed by not requiring an inference if the desired inference has redundant premises. This is crucial in a standard theorem proving derivation, because otherwise the search space grows too large. But we are considering other ways of reducing the search space, and it seems to be less necessary in this context. For instance, it may be more effort to check if a clause is a tautology than to perform a resolution inference involving that tautology. And because of our structure sharing, such an inference may need to be performed anyway, because the literal resolved away may exist in another clause.

We prefer to focus on methods of redundancy which may be performed by local examination of the graph. For instance, after the Basic Paramodulation inference, the right premise may be removed if $s \approx t$ is the only literal in the left premise and the constraint $s\eta \doteq s' \wedge \theta_1\eta$ is a renaming or $\top$. This is called demodulation or simplification. One way to do it is the following. Suppose a replacement hyperedge e from node u is added because of an equation $s \approx t$. Further, suppose that e only contains a rewrite edge from u to v and e has an equational constraint which is just a renaming substitution or $\top$. Then e can be considered as a simplification edge. For most trees, we require that if the tree contains u then it must contain e leading out of u. The only exception we make is when u represent the root of one side of an equation $s \approx t'$ with $t' \leq t$. In that case, we require that no tree containing u has e leading out of it. The reason is that we do not want to let an equation simplify itself or something smaller.

In a resolution inference, the right premise may be removed if A is the only literal in the left premise and the constraint $A \doteq A' \wedge \theta_1\eta$ is a renaming or $\top$. This is called subsumption. We perform subsumption the same way we search for the empty clause as given in the beginning of this section. Whenever a vertex $\Box [\![\varphi]\!]$ is added to $\hat{v}$, then v is considered to represent only the empty clause if φ is a renaming or $\top$. Any node which represents the empty clause is then considered as the empty clause. If this subsumption rule is performed, there is no need to check for the empty clause in the ground case, because the empty clause will automatically be propagated up to one of the nodes labelled by $\vee$.

Now we give the soundness and completeness results. We can define a theorem proving derivation exactly as before. The only difference is that an inference (or several inferences) is performed by adding an edge to the graph and several deletions may be performed by removing an edge from the graph. So S_{i+1} is created from S_i by adding the conclusions of all those inferences or deleting all those clauses.

Theorem 2 *PWD is sound and complete.*

The completeness is because for each inference desired, the edge added to the graph adds the desired conclusion. The soundness is because adding a clause only performs inferences where the conclusion follows from the premises.

5 Complexity

We show that PWD polynomially decides certain classes. For all these classes, only polynomially many edges can be added to the graph, and it only takes polynomial time to perform the next inference at any point. For these complexity arguments we assume that all inferences and redundancy removals are performed by adding an edge to the graph or removing an edge from the graph. Any inference, which can be, will be considered as a simplification or subsumption. We assume that it is possible to calculate the selection rule in polynomial time. We call the inference rules PWD, PWD^+, PWD^- or PWD^{max} to correspond to the selection rules of Basic Paramodulation.

Theorem 3 *PWD polynomially decides $\mathcal{HC}$.*

Proof. Let $S_0 \in \mathcal{HC}$. Let n be the number of literals in S_0. Only the resolution inference rule is applicable to this class, so no positive literals will have outgoing hyperedges. Suppose that u is the root of T_1 which represents $\Gamma \vee A$, which is the left premise of an inference and u_1 represents A in $\Gamma \vee A$. If $\Gamma \vee A \notin S_0$, then there must be some smaller tree with root u such that u_1 represents A. This is true, because the hyperedge from u must have u_1 as a target. Therefore, all necessary inferences involve a member of S_0 and a negative literal. There are only n^2 of these inferences, so only n^2 edges can be added to the graph. The only other thing necessary to check is that the selection rule for all clauses is decidable in polynomial time. In other words, for a given literal, it must be decidable in polynomial time if that literal is selected in some clause. This must be done carefully, because a graph may represent exponentially many clauses. However, it is possible to decide in polynomial time if a literal belongs to a clause, or if a set of literals belong to the same clause. So, for instance, selection rules based on an ordering of the clauses can be decided in polynomial time. A smarter method is to select all negative literals in each goal clause. That saves the time of checking the selection rule, and is very efficient in this method. Note that this argument also holds in the non–ground case. But in that case, it is undecidable if $\Box$ exists with a satisfiable constraint. $\Box$

Theorem 4 *PWD^+ polynomially decides $\mathcal{CN}$.*

Proof. Since the set is saturated by Paramodulation, we know that no inferences are necessary into positive literals. So we only need inferences whose left premise is a clause in S_0. The argument is similar to the argument for $\mathcal{HC}$. $\Box$

Theorem 5 *PWD polynomially decides $\mathcal{UE}$. Therefore Completion is polynomial.*

Proof. Let $S_0 \in \mathcal{UE}$. Let n be the number of symbols in S_0. Since the clauses are unit clauses, the only replacement hyperedges are the ones generated from Basic Paramodulation which contain one rewrite edge or the ones from Equation Resolution which contain no edges. Only one rewrite edge can appear between any two vertices. There can only be n vertices in the graph, hence there are no more than n^2 rewrite edges and therefore no more than n^2+n replacement hyperedges. Since the clauses are ground, each replacement hyperedge is a simplification hyperedge, so each symbol labelled by $\approx$ stands for only one equation. We have to be careful, because the graph may contain exponentially large clauses. However, by an extension of the Patterson-Wegman or Martelli-Montanari unification algorithms [12, 14], the unification could still be performed in linear time. It is only necessary not to write out each clause. $\Box$

Theorem 6 *PWD^- polynomially decides $\mathcal{HCE}$.*

Proof. The left premise is a unit positive equation in every inference, since we perform subsumption. Then the argument is the same as for $\mathcal{UE}$. $\Box$

6 Conclusion

We have presented data structures which create massive structure sharing of a Paramodulation Theorem Prover. This gives theorem proving a polynomial search space for some important subclasses of first order logic with equality. Alternately, this can be viewed as a new inference mechanism for first order logic with equality. This is the view we have taken in the implementations of this procedure which we have begun.

Acknowledgements

I thank the members of the PROTHEO group at INRIA for their comments. Particularly, I thank

Polina Strogova and Christelle Scharff for helpful discussions and a careful reading of this paper. I thank Claude Kirchner for his comments and his backing of this project. Finally, I thank an anonymous referee for useful comments and a careful reading of an early poorly written version of this paper.

References

[1] O. Astrachan and M. Stickel. Caching and lemma use in model elimination theorem provers. In *Proc. 11th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence, vol. 607, pp. 224–238, Berlin, 1992. Springer-Verlag.

[2] L. Bachmair. *Canonical Equational Proofs*. Birkhauser Boston, Inc., Boston MA (1991).

[3] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation Vol. 4, No. 3* (1994) pp. 1–31.

[4] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. In *Proc. 11th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence, vol. 607, pp. 462–476, Berlin, 1992. Springer-Verlag. To appear in *Journal of Information and Computation*.

[5] J. Gallier, P. Narendran, D. Plaisted, S. Raatz, and W. Snyder. Finding Canonical Rewriting Systems Equivalent to a Finite Set of Ground Equations in Polynomial Time. *Journal of Association for Computing Machinery Vol 40, no. 1* (1993) pp. 1–16.

[6] A. Haken. The intractability of resolution. *Theoretical Computer Science 39* (1985) pp. 297–308.

[7] R. Kowalski. A proof procedure using connection graphs. *Journal of the ACM 22* (1975) pp. 572–595.

[8] S. –J. Lee and D. Plaisted. Eliminating duplication with the hyper–linking strategy. *Journal of Automated Reasoning Vol. 9, no. 1* (1992) pp. 25-42.

[9] D. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.

[10] C. Lynch. Local Simplification. In *Constraints in Computational Logic*, Lect. Notes in Computer Science, vol. 845, pp. 3–18, Berlin, 1994. Springer-Verlag.

[11] C. Lynch. Theorem Proving with Equational Horn Clauses: Any Selection Rule is Complete. Tech report 94-R-22, INRIA, 1994.

[12] A. Martelli and U. Montanari. An efficient Unification Algorithm. *ACM Trans. on Prog. Lang and Syst. Vol. 4, no. 2* (1982) pp. 252-282.

[13] R. Nieuwenhuis and A. Rubio. Basic Superposition is Complete. In *Proc. European Symposium on Programming*, Rennes, France (1992).

[14] M. Paterson and M. Wegman. Linear Unification. *Journal of Computer and System Sciences 16* (1978) pp. 158-167.

[15] D. Plaisted. A simplified problem reduction format. *Artificial Intelligence 18* (1982) pp. 227-261.

[16] D. Plaisted. Non–Horn clause logic programming without contrapositives. *Journal of Automated Reasoning 4* (1988) pp. 287-325.

[17] D. Plaisted. The search efficiency of theorem proving strategies. In *Proc. 12th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence, vol. 814, pp. 57–71, Berlin, 1994. Springer-Verlag.

[18] D. Plaisted. The search efficiency of theorem proving strategies: an analytical comparison. Tech report MPI-I-94-233, Max-Planck Institut fuer Informatik, 1994.

[19] D. Plaisted and A. Sattler–Klein. Polynomial Time Completion of Ground Term–Rewriting Systems. Unpublished note.

[20] G.A. Robinson and L. T. Wos. Paramodulation and theorem proving in first order theories with equality. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4* pp. 133–150. American Elsevier, New York, 1969.

[21] W. Snyder. A Fast Algorithm for Generating Reduced Ground Rewriting Systems from a Set of Ground Equations. *Journal of Symbolic Computation 15* (1993) pp. 415-450.

[22] L. Vielle. Recursive Query Processing: the power of logic. *Theoretical Computer Science 69* (1989) pp. 1-53.

[23] D. Warren. Memoing for logic programs. *Communications of the ACM Vol. 35, no. 3* (1992) pp. 94-111.