J. Symbolic Computation (1997) **23**, 23–45

Oriented Equational Logic Programming is Complete

CHRISTOPHER LYNCH[†]

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

(Received 17 May 1995)

We show the completeness of an extension of SLD-resolution to the equational setting. This proves a conjecture of Laurent Fribourg and shows the completeness of an implementation of his. It is the first completeness result for superposition of equational Horn clauses which reduces to SLD resolution in the non-equational case. The inference system proved complete is actually more general than the one of Fribourg, because it allows for a selection rule on program clauses. Our completeness result also has implications for Conditional Narrowing and Basic Conditional Narrowing.

1. Introduction

The combination of logic and functional programming is a popular topic of research. Logic programs are sets of definite Horn clauses (sometimes called program clauses, i.e., clauses containing one positive literal) with goals to be solved represented as negated literals. The SLD-resolution inference rule is a way of solving the goal in a goal-directed fashion. Equational programming is a generalization of functional programming. Equational programs can be represented as sets of equations. Goals are represented as equations. If the set of equations is convergent, a goal can be solved by narrowing (a generalization of rewriting) the goal into an identity. A non-convergent set of equations can be converted into a convergent set by ordering the terms with a well-founded reduction ordering and performing Knuth–Bendix Completion. The ordering on the terms is an important property of Knuth–Bendix Completion. In Completion, an inference rule replaces some instances of a term in an equation by equivalent smaller terms. In some cases, all the instances of a term are replaced, and the equation may be deleted. The combination of logic and equational programming can be represented by equational Horn clauses. In this case, a goal is solved by a combination of SLD-resolution and narrowing. This is the paradigm that is shown complete in this paper.

In logic programming, the property an inference system must satisfy is that whenever G is a goal and θ is a substitution such that $G\theta$ is implied by the program clauses, then some substitution σ must be computed such that θ can be derived from σ. This is generally done by finding a substitution σ that is more general than θ. When the clauses contain

[†] E-mail: `lynch@loria.fr`

equations, the program clauses determine an equational theory E. Therefore, in this case, it is only necessary to find a substitution σ such that σ is more general than θ modulo E. If such substitutions are always computed, we say the inference system is *complete with respect to answer substitutions.* A weaker property, which is often all that is useful in theorem proving, is to show that the empty clause will always be generated from an unsatisfiable set of clauses. This is called *refutational completeness.* A resolution or paramodulation inference system which is complete with respect to answer substitutions is refutationally complete, because the proof that σ is a correct answer is a deduction of the empty clause. In this paper, we save substitutions with each clause. Then when the empty clause is created, the substitution associated with it is an answer substitution. We will prove that our inference system is complete with respect to answer substitutions.

The question of inference systems for equational Horn clauses has been attacked from different angles, by researchers in automated theorem proving and logic programming. However, only Fribourg (1984) gives an inference system that reduces to logic programming in the non-equational case and reduces to a combination of ordered completion (Bachmair, 1991) and narrowing (Kaplan, 1984) in the unit equation case. Fribourg proposes performing superposition among heads of program clauses or between a head of a program clause and a goal equation, but never between the head of a program clause and the condition of another program clause. This system was implemented, however, the author was only able to prove completeness if superposition into variables was allowed and functional reflexivity axioms were added. The author conjectured that completeness still holds if these conditions are dropped. In this paper, we prove that conjecture true. To see an example of this inference system, consider the following set of clauses:

1. $\Rightarrow P(a)$
2. $P(a) \Rightarrow a \approx b$
3. $P(b) \Rightarrow$

where $\approx$ is the equality symbol, $\Rightarrow$ is the implication symbol. We must give an ordering on the terms, because we only need to replace terms by smaller terms. Suppose we choose an ordering such that $a > b$. The last clause $P(b) \Rightarrow$ is the goal. The following is a refutational proof of this set of clauses[†]. The goal clauses are printed in boldface to indicate the structure of the proof.

$$\dfrac{\Rightarrow P(a) \qquad \dfrac{\dfrac{P(a) \Rightarrow a \approx b \qquad \Rightarrow P(a)}{P(a) \Rightarrow P(b)} \qquad \boldsymbol{P(b) \Rightarrow}}{\boldsymbol{P(a) \Rightarrow}}}{\square}$$

One superposition inference is necessary among the heads of the program clauses $P(a) \Rightarrow a \approx b$ and $\Rightarrow P(a)$. The rest of the proof is a series of resolution steps against the goal. We call this method *top down* because no inferences are required among the head of a program clause with the condition of another clause. Notice how this is equivalent to a completion procedure plus a goal-directed proof. The interesting feature of this inference

[†] Without variables, refutational completeness is equivalent to completeness with respect to answer substitutions.

system is its combination of top down and bottom up reasoning. The bottom up part is the completion among the heads of clauses, and the top down part is the goal solving.

Another inference method for equational Horn clauses is the lazy paramodulation method of Snyder and Lynch (1991a), extended to full first-order logic in Snyder and Lynch (1991b). In this method, no paramodulation is needed among definite clauses. However, the inferences are lazy in the sense that the unification problem is saved as a negative equation and not solved immediately. Additionally, ordered paramodulation is not sufficient, and simplification is not allowed. Therefore, this inference system does not reduce to narrowing in the unit case, but it is goal-directed. The following is a proof of the above example in the goal-directed inference system.

$$\dfrac{\Rightarrow P(a) \qquad \dfrac{\Rightarrow P(a) \qquad \dfrac{P(a) \Rightarrow a \approx b \qquad \boldsymbol{P(b)} \Rightarrow}{\boldsymbol{P(a)}, \boldsymbol{P(a)} \Rightarrow}}{\boldsymbol{P(a)} \Rightarrow}}{\Box}$$

All of the inferences are either resolution steps or paramodulation steps against the goal clause. Notice that the first inference performed was not ordered, because $a > b$. In other words, it was necessary to replace a term by a bigger term.

Another inference method for equational Horn clauses is the bottom-up method (Dershowitz, 1991; Nieuwenhuis and Nivela, 1991). This was extended to the first-order case in Bachmair and Ganzinger (1994) and Bachmair *et al.* (1995). In this method, the literals in the body of a definite clause must be solved before the head can be used in an inference with the goal. Therefore, this does not reduce to SLD-resolution in the non-equational case. A bottom up proof of the above example is as follows:

$$\dfrac{\dfrac{\Rightarrow P(a) \qquad \dfrac{\Rightarrow P(a) \qquad P(a) \Rightarrow a \approx b}{a \approx b}}{\Rightarrow P(b)} \qquad \boldsymbol{P(b)} \Rightarrow}{\Box}$$

For every inference, one of the premises is a unit program clause. But the other premise can be a condition of another program clause. The inference against the goal is not until the end of the proof.

A related inference system is one where only a maximal literal in a clause can be used in an inference (Kounalis and Rusinowitch, 1988). This was extended to the first-order case in Rusinowitch (1988), Hsiang and Rusinowitch (1991), Pais and Peterson (1991) and Bachmair and Ganzinger (1990). It is not always possible to choose an ordering such that this reduces to SLD-resolution in the non-equational case. For this strategy, the relationship between refutational completeness and completeness with respect to answer substitutions has been investigated by Nieuwenhuis (1995). None of the known proof techniques for the bottom-up and maximal literal methods can be extended to a method which reduces to SLD-resolution, because they all require an inference to be performed on a negative literal if the maximal literal in the clause is negative. Therefore, it is necessary to develop a new completeness proof technique to prove our completeness result.

We give a general inference method, whose particular cases include the top-down method of Fribourg, the bottom-up method and the maximal literal method, or any combination of these such that different clauses use different strategies. For instance, we may require that we must solve all the negative literals in one program clause before we use the head in an inference. But for another program clause, we may require that the negative literals are not solved until they appear as subgoals of the initial goal. All of these strategies, except for the goal-directed lazy paramodulation strategy, can be encoded by a selection rule that selects one literal from each clause (including the program clauses) to perform inferences on. What we prove is that no matter what selection rule is chosen, the inference system is complete. Therefore we have shown the completeness of each of these inference systems in one proof. However, our main concern is to solve the conjecture of Fribourg, since that is the method which reduces to SLD-resolution and the combination of completion and narrowing. We also show that this strategy can be combined with simplification without losing completeness. However, subsumed clauses and tautologies may not always be removed. But they may be restricted from being involved in inferences with goal clauses. We give conditions showing when they are allowed and examples illustrating why they are not always allowed. Also, we show that our inference system can be combined with a Basic strategy (Hullot, 1980; Bachmair *et al.*, 1992, 1995; Nieuwenhuis and Rubio, 1992, 1995) so that goal clauses are solved without allowing inferences into substitution positions.

Another inference system for equational Horn clauses is Conditional Narrowing (Middledorp and Hamoen, 1994). This inference system is the combination of SLD-resolution and narrowing. Since no completion is involved, this is only complete for certain classes of equational Horn clauses. We show, in this paper, how conditional narrowing can be extended by adding completion so that it is complete for all classes of equational Horn clauses. We also prove it for Basic Conditional Narrowing. We give a comparison of our method with the traditional methods of Conditional Narrowing, and the sense in which we believe our method is an improvement over the other methods.

The form of the paper is as follows. In Section 2, we give the definitions of this paper. At the end of the section, we define the important concept of selection rules. In Section 3, we define a schema for the inference rules used in this paper. Each selection rule as defined in Section 2 gives a particular instance of the inference rule schema. After defining the inference rules, Definition 3.1 shows how to construct a model from a set of Horn clauses. This model contains all the properties needed in the completeness proof. In Section 4, we prove the completeness of each instance of this schema, using the model constructed in Definition 3.1. At the beginning of Section 3, we give an abstract definition to determine when a clause is redundant. In Section 5, we show how the standard deletion rules fit into this abstract framework. In a few cases, the deletion rules do not fit into the frame, and in those cases we exhibit a counterexample showing that those deletion rules are not complete. In Section 6, we show the relationship between our method and the traditional methods of Conditional Narrowing. Finally, in the conclusion we summarize our results and give some ideas for future enhancements of the technique.

2. Preliminaries

Our setting is Horn clauses with equality predicates. We use mostly standard notions of rewriting (Dershowitz and Jouannaud, 1990) and only define notions that may not be standard.

Definition 2.1. *An atom is a predicate applied to some terms. A special case is the equality predicate $\approx$ which is a binary predicate represented in infix notation. A* model *is* *a set of ground atoms. If M is a model and A is a ground atom, we say that M* implies *A, and write $M \models A$ if M implies A in all equational theories. If Γ is a set of ground atoms, then $M \models \Gamma$ if $M \models A$ for all $A \in \Gamma$*[†]*. If S and T are sets of equations, we say that $S \equiv T$ if for all equations A, $S \models A$ if and only if $T \models A$. If E is a set of equations and s and t are terms, then $s \equiv_E t$ means that $E \models s \approx t$. For equations $s \approx t$ and $s' \approx t'$, we define $(s \approx t) \equiv_E (s' \approx t')$ to mean that $s \equiv_E s'$ and $t \equiv_E t'$.*

We define ground clauses and substitutions.

Definition 2.2. *A* substitution *is a mapping from variables to terms which is almost everywhere the identity. We define* $\mathrm{Dom}(\sigma) = \{x \mid x\sigma \neq x\}$*. We identify a substitution with its homomorphic extension. The composition $\sigma\eta$ of substitutions σ and η is defined so that for all terms t, $t\sigma\eta = (t\sigma)\eta$. Let V be a set of variables and E be a set of equations. We say that $\sigma \leq_E \theta|_V$ if there is a substitution η such that, for all $x \in V$, $E \models x\sigma\eta \approx x\theta$. If E is missing it is assumed to be the empty set. If V is missing it is assumed to be the set of all variables. We say $\sigma \equiv_E \theta|_V$ if and only if $E \models x\sigma \approx x\theta$ for all $x \in V$. We say $\sigma = \theta|_V$ if $x\sigma = x\theta$ for all $x \in V$. The substitution σ is a* unifier *of terms s and t if $s\sigma = t\sigma$. σ is a* most general unifier *of s and t (written $mgu(s,t)$) if σ is a unifier of s and t, and for all unifiers θ of s and t, $\sigma \leq \theta$.*

We assume we have a procedure which produces idempotent *mgu*'s (i.e., for all variables x, x is not a proper subterm of $x\sigma$).

Since we prove completeness with respect to answer substitutions, we separate a clause from its substitution. A ground instance is then just a further instantiation of the substitution (the substitution is still kept separate)[‡]. We warn the reader that two terms can be considered to be different terms, even if they become the same when their substitutions are applied.

Definition 2.3. *An* equational Horn clause *C is a pair containing a clause and a substitution, of the form $\Gamma \Rightarrow \Delta [\![\sigma]\!]$, where Γ is a set of atoms and Δ is a set of atoms, containing zero or one atom. When the substitution σ is not important we simply write the above clause as $\Gamma\sigma \Rightarrow \Delta\sigma$. If Δ contains no atoms then C is called a* goal clause *and is written as $\Gamma\sigma \Rightarrow$. If Δ contains one literal A then C is called a* program clause *and is written as $\Gamma\sigma \Rightarrow A\sigma$. If Γ is empty and Δ contains the literal A then we write it as $\Rightarrow A\sigma$ and call it a* fact*. If both Γ and Δ are empty, then we write $\Box$ and call it the* empty clause*. If $\Gamma\sigma \Rightarrow \Delta\sigma$ is a ground clause, we say that $M \models \Gamma\sigma \Rightarrow \Delta\sigma$ if and only if $M \not\models \Gamma\sigma$ or there exists a $B \in \Delta\sigma$ such that $M \models B$.*

We have presented clauses as a combination of an unconstrained clause and a substitution. The substitution can be considered as a constraint on the instances of a clause (Kirchner *et al.*, 1990). The most general unifier $\sigma = mgu(s,t)$ is an equational constraint, which is just a simplification of the equational constraint $s = t$. We choose to represent it as the most general unifier to make it clear how the answer substitutions are read off from the constraint, but it could just as easily be represented as the equation

[†] This is trivially true if Γ is empty.

[‡] Although, it is often written as if it were applied, when it does not matter in the context.

$s = t$. There are many recent results on constrained deduction. Viewing most general unifiers as equational constraints illustrates how this paper fits into that framework.

Definition 2.4. *The substitution σ is a* grounding substitution of $C\llbracket\theta\rrbracket$ *if $C\theta\sigma$ contains no variables. If $C\llbracket\theta\rrbracket$ is a clause then $Gr(C\llbracket\theta\rrbracket) = \{C\llbracket\theta\sigma\rrbracket \mid \sigma$ is a grounding substitution of $C\theta\}$. If D is in $Gr(C\llbracket\theta\rrbracket)$, then D is an instance of $Gr(C\llbracket\theta\rrbracket)$. If S is a set of clauses, then $Gr(S) = \bigcup_{C\in S} Gr(C)$. We consider clauses $C\llbracket\theta\rrbracket$ and $D\llbracket\sigma\rrbracket$ to be* identical *if they are goal clauses and $C\llbracket\theta\rrbracket = D\llbracket\sigma\rrbracket$, or if they are program clauses and there is a* renaming substitution *η (i.e., for all variables x, $x\eta$ is a variable and η is injective) such that $C\eta = D$ and $C\theta\eta = D\sigma$.*

Let V be the set of variables. We will divide V into a set V_G of *goal variables* and a set V_P of *program variables.* The goal variables are the variables which appear in the initial goal clauses. The program variables are the variables which appear in the initial program clauses, plus any fresh variables that are created. Variables in program clauses are renamed before an inference is performed.

To simplify the proofs, we encode each of the clauses in a new signature, so that all literals are equality literals. This requires adding a new constant $\top$. Each non-equational atom A is encoded as the equation $A \approx \top$. Predicate symbols in A become function symbols in this encoding. The purpose for this encoding is that we now only need to deal with equations, which makes the completeness proof easier to read. Therefore, we don't present the resolution inference rule, because it is encoded by a goal paramodulation and equation resolution inference.

A reduction ordering on terms must be given, which is then extended to equations. We define some reducibility notions based on that ordering.

Definition 2.5. *Let $\prec_r$ be a reduction ordering on terms and non-equational atoms, total on ground terms. We identify $\prec_r$ with its multiset extension, extended to equations by considering an equation $s \approx t$ as the multiset $\{s, t\}$.*

Definition 2.6. *Let $A\llbracket\theta\rrbracket$ be an equation and $A\llbracket\theta\sigma\rrbracket$ be a ground instance of $A\llbracket\theta\rrbracket$. Let R be a rewrite system. We say that $A\llbracket\theta\sigma\rrbracket$ is* substitution reduced by R *if there is no equation $s \approx t$ in R such that $s \approx t \prec_r A\theta\sigma$, $s \succ_r t$, and s is a subterm of $x\theta\sigma$ for some variable x in A. We define $\theta\downarrow_R$ as the substitution θ' such that* $\mathrm{Dom}(\theta) = \mathrm{Dom}(\theta')$, *for all $x \in \mathrm{Dom}(\theta)$, $x\theta' \equiv_R x\theta$, and there is no t such that $t \equiv_R x\theta'$ and $t \prec_r x\theta'$.*

Next we define a set S of ground equations to be *left-reducible* if some equation in S reduces the maximum side of some other equation. The word "left" refers to the fact that if an equation is written as a rewrite rule, then the maximum side appears on the left.

Definition 2.7. *Let $u \approx v$ be a ground equation and S be a set of ground equations. We say that $u \approx v$ is* left-reducible by S *if $u \succ_r v$ and there is an equation $s \approx t \in S$ such that s is a subterm of u, $s \succ_r t$ and $(s \approx t) \prec_r (u \approx v)$. If $u \approx v$ is not left-reducible by S then it is* left-irreducible by S.

Definition 2.8. *A set of equations R is* reduced *if, for each equation $u \approx v$ in R, $u \approx v$ is substitution reduced by R and $u \approx v$ is left-irreducible by R.*

Notice that a reduced set of equations is always convergent. This is because the largest side of each equation is reduced by the other equations.

A selection rule is a function from a clause to exactly one literal in that clause. Only selected literals must be used in inferences.

DEFINITION 2.9. *A* selection rule *is a function Sel from the set of all clauses to the set of all literals such that* $Sel(C [\![\theta]\!]) \in C [\![\theta]\!]$ *for all clauses* $C [\![\theta]\!]$. *If* $Sel(C [\![\theta]\!]) = L [\![\theta]\!]$, *we say that* L *(or* $L [\![\theta]\!]$*) is* selected in $C [\![\theta]\!]$ *or that* $C [\![\theta]\!]$ selects L *(or* $L [\![\theta]\!]$*).*

In Section 3, we show how the selection rule is used in inferences and how it can be used to encode the maximal literal strategy, we select a maximal literal in each clause. To encode the bottom-up method, we select one negative literal in each clause containing one, and a positive literal otherwise. This guarantees that each inference will contain a fact, since each inference must contain a clause with a positive selected literal and only facts will have a positive literal selected. To encode the top-down strategy of Fribourg, we select the positive literal in each program clause and one negative literal in each goal clause. See Section 3 for the details.

3. The Inference Rules

In this section, we give the inference rules.

There are two paramodulation inference rules. *Superposition* is a paramodulation among the heads of program clauses and *Goal Paramodulation* is a paramodulation into a goal clause or the condition of a program clause. It is called Goal Paramodulation because it is used to solve a negative literal, which is a current or potential goal. In the strategy which always selects a positive literal of a program clause, it is only used to solve a current goal, because only goal clauses have a negative literal selected. The two inference rules have exactly the same form; the only difference is whether a negative or positive literal is paramodulated into.

SUPERPOSITION

$$\frac{\Gamma \Rightarrow s \approx t [\![\theta_1]\!] \qquad \Delta \Rightarrow u[s'] \approx v [\![\theta_2]\!]}{\Gamma, \Delta \Rightarrow u[t] \approx v [\![\sigma \wedge \theta_1 \wedge \theta_2]\!]}.$$

GOAL PARAMODULATION

$$\frac{\Gamma \Rightarrow s \approx t [\![\theta_1]\!] \qquad u[s'] \approx v, \Delta \Rightarrow \Pi [\![\theta_2]\!]}{u[t] \approx v, \Delta, \Gamma \Rightarrow \Pi [\![\sigma \wedge \theta_1 \wedge \theta_2]\!]}.$$

For both of the above inference rules, we have the following conditions.

1. $\sigma = mgu(s\theta_1, s'\theta_2)$,
2. s' is not a variable,
3. $s \approx t [\![\theta_1]\!]$ is selected in $\Gamma \Rightarrow s \approx t [\![\theta_1]\!]$,
4. $u[s'] \approx v [\![\theta_2]\!]$ is selected in its clause,
5. $s\theta_1\sigma \not\prec_r t\theta_1\sigma$, and
6. $u[s']\theta_2\sigma \not\prec_r v\theta_2\sigma$.

To complete the inference system, we need an inference rule, called *Equation Resolution* which is applied to get an answer substitution for a solved equation.

EQUATION RESOLUTION

$$\frac{s \approx t, \Delta \Rightarrow \Pi \llbracket \theta \rrbracket}{\Delta \Rightarrow \Pi \llbracket \sigma \wedge \theta \rrbracket}$$

where

1. $\sigma = mgu(s\theta, t\theta)$, and
2. $s \approx t \llbracket \theta \rrbracket$ is selected in $s \approx t, \Delta \Rightarrow \Pi \llbracket \theta \rrbracket$.

Resolution can be encoded by Paramodulation and Equation Resolution. Paramodulation into non-equality predicates is represented here by its encoding.

We now show how the inference systems mentioned in the introduction are encoded by the selection rule.

The inference system of Kounalis and Rusinowitch (1988) only allows inferences involving maximal literals. This is expressed by modifying the above inference rules so that condition number 3 is modified to read: $s \approx t \llbracket \theta_1 \rrbracket$ is maximal in $\Gamma \Rightarrow s \approx t \llbracket \theta_1 \rrbracket$, and condition 4 is modified to read: $u[s'] \approx v \llbracket \theta_2 \rrbracket$ is maximal in its clause. Furthermore, condition 2 of Equation Resolution is modified to read: $s \approx t \llbracket \theta \rrbracket$ is maximal in $s \approx t, \Delta \Rightarrow \Pi \llbracket \theta \rrbracket$. This is encoded by the selection rule that selects a maximal literal in each class. In fact the encoding gives a more restrictive inference system because a clause may have more than one maximal literal.

The bottom-up inference system of Dershowitz (1991) and Nieuwenhuis and Nivela (1991) only allows an inference involving the head of a program clause if the body of that clause is empty. We encode this rule by selecting a negative literal in every clause that has one. It can be expressed by the given inference rules if we require that $\Gamma = \emptyset$ and $\Delta = \emptyset$ for the Superposition inference rule, and that $\Gamma = \emptyset$ for the Goal Paramodulation inference rule. That means that conditions 3 and 4 are not necessary for the Superposition inference rule. For the Goal Paramodulation inference rule condition 3 is not necessary.

Next, we consider the top down inference system, which was conjectured to be complete in Fribourg (1984), which does not allow any inferences into the body of a program clause. We express it in our system by setting $\Pi = \emptyset$ for the Goal Paramodulation and Equation Resolution inference rule. We remove conditions 3 and 4 from the Superposition inference rule, and remove condition 3 from the Goal Paramodulation inference rule. It is encoded by the selection rule that selects a positive literal in any clause containing one.

We use the selection rule to build a model of the set of program clauses. The process is similar to the model construction process in completeness proofs of SLD-resolution. We create the model in levels. On level 0 we add the facts. Then on each succeeding level we add all the consequences of the preceding level. But since we have equalities, we must also add equational consequences. So the process is divided up into two steps. For each level n we create L_n which contains all the equations at the head of a program clause whose conditions are true at the previous level. Then we create M_n which contains all the equational consequences of L_n. In this manner, we eventually add every equation which is implied by the program clauses. We can use these levels to create an ordering on the equations, so that as in the proof of SLD-resolution each goal clause $\Gamma \Rightarrow$ where Γ

is implied by the model and can be involved in an inference which yields a smaller goal clause, as we will see in Section 4. In order for this to be the case, we only add equations to M_n which are implied by some convergent subset R of L_n. Furthermore, the added equation must be substitution reduced by R. Finally, in order to show that simplifications reduce the size of an equation, we require that this R implies all the facts, since these are the simplifiers. But we must be sure that this process eventually adds all the equations which follow from the program clauses. For instance an equation A might follow from L_n but not from a convergent subset of L_n. Therefore A will not be in M_n. However, since A is an equational consequence of the program clauses, we must prove that A is added later on in the model construction.

Definition 3.1. *Let S be a set of ground equational Horn clauses. For $n \geq 0$, let L_n and M_n be sets of equations such that $A \in L_n$ if and only if $A \in L_m$ for some $m < n$ or there is a clause $C = (\Gamma \Rightarrow A) \in Gr(S)$ such that A is selected in C and $\Gamma \subseteq M_{n-1}$. We say that C* produces *A in L_n, or that C is* productive. *Let M_n be the set of all equations B such that there is a set of equations $R \subseteq L_n$ such that $R \models L_0$, $R \models B$, R is reduced and B is substitution reduced by R. Define $L_\infty = \bigcup_{n \geq 0} L_n$, and $M_\infty = \bigcup_{n \geq 0} M_n$.*

If n is the smallest number such that $A \in M_n$, we say $level(A) = n$. If $A \notin M_\infty$, we say $level(A) = \infty$. We define an ordering $\prec_l$ on atoms by $A \prec_l B$ if and only if $level(A) < level(B)$ or else $level(A) = level(B)$ and $A \prec_r B$[†].

We extend the orderings so they apply to atoms and clauses.

Definition 3.2. *We extend $\prec_l$ to negative literals so that $A \prec_l \neg B$ if $level(A) \preceq_l level(B)$ and $\neg A \prec_l B$ if $level(A) \prec_l level(B)$. We identify $\prec_l$ with its multiset extension. Then a clause $\Gamma \Rightarrow \Delta$ is identified with the set $\Delta \cup \{\neg A \mid A \in \Gamma\}$. We extend the $\prec_r$ ordering to ground clauses in the following way. If $C = (\Gamma \Rightarrow A)$ and $D = (\Delta \Rightarrow B)$ then $C \prec_r D$ if and only if $A \prec_r B$ or else $A = B$ and $\Gamma \prec_l \Delta$.*

We illustrate the last two definitions on the following set of clauses.

1. $\Rightarrow P(a, b)$
2. $P(x, b) \Rightarrow Q(x, b)$
3. $Q(a, b) \Rightarrow a \approx b$
4. $Q(b, a) \Rightarrow$

with a lexicographic path ordering based on the precedence $P \succ_r Q \succ_r a \succ_r b$.

In the sequel, we will only apply the model construction process to a set of clauses which has been saturated by Superposition. If we use a selection rule that selects the head of every program clause, then there is one inference among the program clauses in this example:

$$\frac{Q(a,b) \Rightarrow a \approx b \qquad \Rightarrow P(a,b)}{Q(a,b) \Rightarrow P(b,b)}.$$

So for Definition 3.1, the set S is $\{\Rightarrow P(a,b), P(x,b) \Rightarrow Q(x,b), Q(a,b) \Rightarrow a \approx b, Q(b,a) \Rightarrow, Q(a,b) \Rightarrow P(b,b)\}$. The goal clause $Q(b,a) \Rightarrow$ is included in the set, even

[†] The $\prec_l$ ordering depends on S. If it is necessary to know what S is, we will say $A \prec_l B$ *with respect to* S.

though goal clauses are not used in the model construction. The set of all ground instances of S, called $Gr(S)$, is $\{\Rightarrow P(a,b), P(x,b) \Rightarrow Q(x,b) \llbracket x \mapsto a \rrbracket, P(x,b) \Rightarrow Q(x,b) \llbracket x \mapsto b \rrbracket, Q(a,b) \Rightarrow a \approx b, Q(b,a) \Rightarrow, Q(a,b) \Rightarrow P(b,b)\}$. The clause $P(x,b) \Rightarrow Q(x,b) \llbracket x \mapsto a \rrbracket$ will sometimes be referred to as $P(a,b) \Rightarrow Q(a,b)$ in situations when the border between the clause and the substitution is irrelevant.

We have assumed that all non-equational atoms are encoded by equations. So in this example, any non-equational atom A can be read as $A \approx \top$. However, for readability, we write it in its unencoded form.

Definition 3.1 puts all the facts into L_0, therefore $L_0 = \{P(a,b)\}$. We defined M_0 to be the set of all equations B which are implied by a convergent subset R of L_0 such that B is substitution reduced by R. For technical reasons, so as to show that simplification really is a case of redundancy, we also require that $R \models L_0$. In this case, we can let $R = L_0$ and $P(a,b)$ trivially follows from R. Therefore $M_0 = L_0 = \{P(a,b)\}$. In reality, M_0 is a set of ground instances. So it includes, for example, $P(x,y) \llbracket x \mapsto a, y \mapsto b \rrbracket$, because a and b are irreducible by M_0.

Next, L_1 is the set of all equations which appear as the head of a clause $C \in Gr(S)$ whose conditions are in M_0. So, $L_1 = \{P(a,b), Q(a,b)\}$. $Q(a,b)$ is added to the set because of the clause $P(a,b) \Rightarrow Q(a,b)$. We can now set $R = L_1$ and see that $M_1 = L_1$. The set L_2 is $\{P(a,b), Q(a,b), a \approx b, P(b,b)\}$, because of the clauses $Q(a,b) \Rightarrow a \approx b$ and $Q(a,b) \Rightarrow P(b,b)$. In this case, we cannot set $R = L_2$, since L_2 is not convergent. We can set $R = \{P(a,b), a \approx b, P(b,b)\}$. This allows us to say that $P(b,a)$ and $P(a,a)$ are in M_2. We can set $R = \{Q(a,b\}$ to show that $Q(a,b) \in M_2$. But $Q(b,a)$ and $Q(a,a)$ are not in M_2 even though they are logically implied by L_2, because neither of them are implied by a convergent subset of L_2. Therefore, $M_2 = \{P(a,b), Q(a,b), a \approx b, P(b,b), P(b,a), P(a,a)\}$. Finally, $L_3 = \{P(a,b), Q(a,b), a \approx b, P(b,b), Q(b,b)\}$, which is convergent. So, we can set $R = \{P(b,b), Q(b,b), a \approx b\}$ and then $M_3 = \{P(a,b), Q(a,b), a \approx b, P(b,b), P(b,a), P(a,a), Q(b,b), Q(b,a), Q(a,a)\}$. Additionally, $L_4 = L_3$, which means no further equations are added to the model.

The model construction gives us an ordering on the atoms of: $P(a,b) \prec_l Q(a,b) \prec_l a \approx b \prec_l P(b,b) \prec_l P(b,a) \prec_l P(a,a) \prec_l Q(b,b) \prec_l Q(b,a) \prec_l Q(a,a)$. In the next section, we will prove that L_∞ is convergent. This is true in this example, because $L_\infty = L_3$, which we have already noted is convergent. We also prove in the next section that if $\Gamma \Rightarrow$ is a goal clause such that $L_\infty \models \Gamma$, then there is an inference of $\Gamma \Rightarrow$ with a clause of S such that a new smaller goal clause is produced.

We show the proof of $Q(b,a) \Rightarrow$ in the above example. There is an inference with clause 3 which produces $Q(a,b), Q(b,b) \Rightarrow$. This is smaller than the original goal, because $\mathit{level}(Q(a,b)) = 1 < 3 = \mathit{level}(Q(b,a))$, and $\mathit{level}(Q(b,b)) = \mathit{level}(Q(b,a))$ but $Q(b,b) \prec_r Q(b,a)$. Next there is an inference involving clause 2 with subgoal $Q(a,b)$. The resulting subgoal is $P(x,b), Q(b,b) \Rightarrow \llbracket x = a \rrbracket$. This is smaller than the previous subgoal, because $\mathit{level}(P(a,b)) = 0$ and $\mathit{level}(Q(a,b)) = 1$. Next, an inference with clause 1 eliminates $P(a,b)$ and results in the new subgoal $Q(b,b) \Rightarrow$. After that, an inference with clause 2 again gives us $P(x,b) \llbracket x = b \rrbracket$, which is smaller because $\mathit{level}(P(b,b)) = 2$ and $\mathit{level}(Q(b,b)) = 3$. Then an inference with $Q(a,b) \Rightarrow P(b,b)$ gives us $Q(a,b) \Rightarrow$. Again we have reduced the level of the subgoal. Finally, we apply the same steps to $Q(a,b)$ as we did above, to eventually eliminate it.

4. Completeness

We define the notion of redundancy used in this paper. It is much more complicated than the notion of Bachmair and Ganzinger (1994), which says a clause is redundant if it is implied by smaller clauses. This complexity is because the notion of redundancy must be more restrictive than the usual notion. In Section 5, we show counterexamples which show it is necessary to be this restrictive in order to preserve completeness.

Definition 4.1. *A clause D is S–*redundant w.r.t. *$D_1, \ldots, D_n$ if, for all ground instances C of D, for all i, there exists a C_i which is an instance of D_i such that, $C_i \preceq_l C$ with respect to S and $C_1, \ldots, C_n \models C$. Also*

1. *If C is a goal clause then C is of the form $C' [\![\theta\eta]\!]$ and we require that exactly one C_i is a goal clause and it is of the form ${C_i}' [\![\sigma\tau]\!]$ such that* $\mathrm{Dom}(\theta) \subseteq \mathrm{Dom}(\sigma)$, *and for all variables $x \in \mathrm{Dom}(\theta)$, $x\theta = x\sigma$ and $x\theta\eta = x\sigma\tau$.*
2. *If C is a program clause, then C is of the form $\Gamma \Rightarrow A$, and we require that*
   (a) *each C_i is a program clause of the form $\Gamma_i \Rightarrow A_i$,*
   (b) *$A_1, \ldots, A_n \models A$,*
   (c) *for each i, $C_i \preceq_r C$, and*
   (d) *for all R, C_i is substitution reduced by R whenever C is substitution reduced by R.*

*A clause D is S–*redundant in *T if there exist clauses $D_1, \ldots, D_n \in T$ such that D is S-redundant w.r.t. $D_1, \ldots, D_n$. A clause D is* strictly S-redundant in *T if there exist clauses $D_1, \ldots, D_n \in T$ such that D is redundant w.r.t. $D_1, \ldots, D_n$ and there is no D_i such that D_i is redundant w.r.t. D, $D_1, \ldots, D_{i-1}, D_{i+1}, \ldots, D_n$.*

This definition allows for most of the redundancy notions used in practice, but not all of them. For goal clauses, tautology deletion and simplification are allowed, and subsumption is allowed when possible after a resolution with a unit clause. More general cases of subsumption are not allowed if information is lost about the answer substitution.

The most important inference rule in practice is simplification. Fortunately, it is always possible to simplify a program clause and remove the simplified clause. This is not immediately obvious by the above definition. When an equation is simplified, the result is a new equation which is smaller with respect to $\prec_r$, but it is not obvious that is is smaller with respect to $\prec_l$. However, it will be smaller with respect to $\prec_l$, because we have built that into the definition of the ordering. When the model is constructed in Definition 3.1, an equation A is only added to M_n if there is some $R \subseteq L_n$ such that $R \models A$ and $R \models L_0$. Recall that when an equation is simplified, it must be simplified by a unit equation. Therefore, if A is at level n, then a simplified version of A must have been added to the model at or before level n. We refer the reader to Section 5 for a more detailed discussion of redundancy.

Other redundant program clauses may not always be removed. Tautology removal is possible in some cases. For instance, clauses of the form $\Gamma \Rightarrow t \approx t$ may be removed. Clauses of the form $A, \Gamma \Rightarrow A$ may not be removed, because this violates condition 2(b) in the above definition. For subsumption, a clause C may be removed if C is subsumed by a clause D where $D \subseteq C$, but not in general. For instance, the clause $P(a, a) \Rightarrow P(a, b)$ cannot be removed in the presence of the clause $P(x, x) \Rightarrow P(x, b)$ because this violates condition 2(d) of the definition of redundancy. In any case, clauses which are redundant in the usual sense but not in our sense may by forbidden to be used in inferences involving a goal clause, as given in the next definition. Again, the reader is referred to Section 5 for a more detailed discussion of redundancy.

Now we give some facts about redundancy to show that redundancy is not affected by adding new clauses or deleting redundant clauses.

PROPOSITION 4.1. *Let S, T and T' be sets of clauses such that $T \subseteq T'$. Suppose that D is S-redundant in T. Then D is S-redundant in T'.*

PROOF. If D is S-redundant in T, then there exist $D_1, \ldots, D_n \in T$ such that D is S-redundant w.r.t. $D_1, \ldots, D_n$. Since, $T \subseteq T'$, $D_1, \ldots, D_n \in T'$. So D is S-redundant in T'. □

PROPOSITION 4.2. *Let S be a set of clauses. Suppose that D is redundant w.r.t. $D_1, \ldots, D_n$ and D_1 is redundant w.r.t. $E_1, \ldots, E_m$. Then D is redundant w.r.t. $E_1, \ldots, E_m, D_2, \ldots, D_n$.*

PROOF. Let C be an instance of D. Then for each i, $1 \le i \le n$ there is an instance C_i of D_i such that $C_i \preceq_l C$ with respect to S and $C_1, \ldots, C_n \models C$. Also, for each j, $1 \le j \le m$, there is an instance F_j of E_j such that $F_j \preceq_l C_1$ with respect to S and $F_1, \ldots, F_m \models C_1$. By the transitivity of $\preceq_l$, we know that $F_j \preceq_l C$ for all j. Also, $F_1, \ldots, F_m, C_2, \ldots, C_n \models C$. We must show that one of the two cases in the definition of S-redundancy applies.

1. If C is a goal clause then C is of the form $C' [\![\theta\eta]\!]$ and there is a goal clause $C_i \in \{C_1, \ldots, C_n\}$ of the form $C_i' [\![\sigma\tau]\!]$ such that $\mathrm{Dom}(\theta) \subseteq \mathrm{Dom}(\sigma)$, and for all variables $x \in \mathrm{Dom}(\theta)$, $x\theta = x\sigma$ and $x\theta\eta = x\sigma\eta$. We must show that one of $F_1, \ldots, F_m, C_2, \ldots, C_n$ also has this property. If $C_i \in \{C_2, \ldots, C_n\}$, then C_i is in the set. Otherwise, suppose that $C_i = C_1$. Then there is a goal clause $F_j \in \{F_1, \ldots, F_m\}$ of the form $F_j' [\![\sigma'\tau']\!]$ such that $\mathrm{Dom}(\sigma) \subseteq \mathrm{Dom}(\sigma')$, and for all variables $x \in \mathrm{Dom}(\sigma)$, $x\sigma = x\sigma'$ and $x\sigma\tau = x\sigma'\tau'$. This implies that $\mathrm{Dom}(\theta) \subseteq \mathrm{Dom}(\sigma')$, and for all variables $x \in \mathrm{Dom}(\theta)$, $x\theta = x\sigma'$ and $x\theta\eta = x\sigma'\tau'$. Therefore F_j is the required clause.
2. If C is a program clause, then C is of the form $\Gamma \Rightarrow A$ and
   (a) each C_i is a program clause of the form $\Gamma_i \Rightarrow A_i$,
   (b) $A_1, \ldots, A_n \models A$,
   (c) for each i, $C_i \preceq_r C$, and
   (d) for all R, C_i is substitution reduced by R whenever C is substitution reduced by R.

   We also know that

   (a) each F_j is a program clause of the form $\Delta_j \Rightarrow B_j$,
   (b) $B_1, \ldots, B_m \models C_1$,
   (c) for each j, $F_j \preceq_r C_1$, and
   (d) for all R, F_j is substitution reduced by R whenever C_1 is substitution reduced by R.

   This implies that

(a) each F_j is a program clause of the form $\Delta_j \Rightarrow B_j$,
(b) $B_1, \ldots, B_m, A_2, \ldots, A_n \models C$,
(c) for each j, $F_j \preceq_r C$, and
(d) for all R, F_j is substitution reduced by R whenever C is substitution reduced by R.

Therefore D is redundant w.r.t. $E_1, \ldots, E_m, D_2, \ldots, D_n$. $\Box$

Proposition 4.3. *Let S, T and T' be sets of clauses such that $T \subseteq T'$, and all members of $T' \setminus T$ are S-redundant in T'. If D is S-redundant in T', then D is S-redundant in T.*

Proof. If D is S-redundant in T', then there exist $D_1, \ldots, D_n \in T'$ such that D is S-redundant w.r.t. $D_1, \ldots, D_n$. We know that each D_i is either in T or is S-redundant in T. By applying the previous proposition several times, we see that D is S-redundant in T. $\Box$

We also define redundant inferences.

Definition 4.2. *An inference*

$$\frac{C_1}{C\sigma}$$

is S–redundant in T if $C_1\sigma$ is strictly S–redundant in T.

An inference

$$\frac{C_1 \qquad C_2}{C\sigma}$$

is S–redundant in T if $C_1\sigma$ or $C_2\sigma$ is strictly redundant in T, or if the inference is a resolution or goal paramodulation inference and $C_1\sigma$ is not productive in T. We say a clause is productive in T *if some instance is productive in some set of clauses containing $Gr(T)$.*

This definition implies that if a program clause C is implied by smaller clauses, then a resolution or goal paramodulation inference involving C is redundant, even though C may not be redundant.

In this section we prove the completeness of the inference system given in this paper. The inference system is sound. To prove completeness, we use the definition of a fair theorem proving derivation, meant to model an automated theorem prover (from Bachmair, 1991).

Definition 4.3. *Let $S_0, S_i, S_2, \ldots$ be a (finite or countably infinite) sequence of sets of clauses. A clause C is said to be* persisting *if there exists some j such that for every $k \geq j$, there exists an identical clause $C' \in S_k$. The set of all persisting clauses, denoted S_∞, is called the* limit *of the derivation. A clause C in some S_i is* redundant *if it is S_∞-redundant.*

The sequence $S_0, S_1, S_2, \ldots$ is called a theorem proving derivation *if each set S_{i+1} can be obtained from S_i by adding a clause which is a consequence of S_i or by deletion of a clause C that is redundant in $S_i \setminus C$.*

A set of clauses S is saturated *if every inference applied to clauses in S is redundant in S. A theorem proving derivation is called* fair *if S_∞ is saturated.*

From Propositions 4.1 and 4.3, it follows that if C is redundant in some S_i then C is redundant in S_∞. In the sequel, we will use the notion of redundancy with respect to S_∞.

Now we present the completeness theorem of our inference rules, preceded by some necessary lemmas. The first lemma shows that the goal paramodulation and equation resolution inference rules reduce the size of the right premise, w.r.t. both orderings on clauses.

LEMMA 4.1. *Let $S = S_\infty$ for some fair theorem proving derivation, from which the sequences $L_0, \ldots$ and $M_0, \ldots$ have been constructed. Let $C = (\Gamma \Rightarrow \Pi)$ be a non-redundant clause in $Gr(S)$ with a negative selected literal, such that $M_\infty \models \Gamma$. Then there is a clause $D = (\Lambda \Rightarrow \Pi)$ in $Gr(S)$ or redundant in $Gr(S)$, such that $\Lambda \prec_l \Gamma$ and $M_\infty \models \Lambda$. Furthermore, if θ is the substitution part of C and σ is the substitution part of D, then $\sigma = \theta|_{V_G}$.*

PROOF. C must be of the form $A, \Gamma' \Rightarrow \Pi$ where A is selected. Suppose that A is of the form $t \approx t$. Then there must be a ground instance of an equation resolution inference.

$$\frac{t \approx t, \Gamma' \Rightarrow \Pi}{\Gamma' \Rightarrow \Pi}.$$

Then $\Gamma' \Rightarrow \Pi$ has the desired properties, and it is either an element of $Gr(S)$ or is redundant in $Gr(S)$.

So suppose A is of the form $u \approx v$, where $u \succ_r v$. If $u \approx v$ is at level m then there must be a reduced set $R' \subseteq L_m$ such that $R' \models u \approx v$ and $u \approx v$ is substitution reduced by R'. So $u \approx v$ has a rewrite proof in L_m. Let $s \approx t$ be the first step of the rewrite proof into u So $s \approx t$ is produced by some member $\Delta \Rightarrow s \approx t$ of S, where $s \approx t$ is substitution reduced by R'. Then there is a ground instance of a goal paramodulation inference in S.

$$\frac{\Delta \Rightarrow s \approx t \qquad u[s] \approx v, \Gamma' \Rightarrow \Pi}{u[t] \approx v, \Delta, \Gamma' \Rightarrow \Pi}.$$

We know that $u[t] \approx v$ is substitution reduced by R' , because $u[s] \approx v$ and $s \approx t$ are substitution reduced by R'. The only potential problem would be if s was of the form $x\theta$ for some variable x, but then that variable would not appear in $u[t] \approx v$. By definition of $\prec_l$, $(u[t] \approx v, \Delta, \Gamma') \prec_l (u[s] \approx t, \Gamma')$, because Δ is at some level less than m and $u[t] \approx v$ is at level m and smaller than $u[s] \approx v$ w.r.t. $\prec_r$. Therefore $u[s] \approx v$ has been replaced with smaller things w.r.t. $\prec_l$. The equation $u[t] \approx v$ is in M_∞. Also, note that the substitution part of $u[t] \approx v, \Delta, \Gamma' \Rightarrow \Pi$ is the same as the substitution part of C, except that it may have some additional program variables. Therefore $u[t] \approx v, \Delta, \Gamma' \Rightarrow \Pi$ has the desired properties. It is either an element of $Gr(S)$ or redundant in $Gr(S)$. $\Box$

In order to preserve completeness, we need to show that the model L_∞, which we constructed, is convergent. We next define how to create a reduced version of a set of equations E. If that reduced version is logically equivalent to E, that will imply that E is convergent.

DEFINITION 4.4. *Let E be a ground set of equations. Define $RedVar(E) = \{A \in E \mid A$ is left-irreducible by R and A is substitution reduced by $R\}$.*

This definition is well defined, because the definitions of *reduced* and *left irreducible* only use smaller equations.

PROPOSITION 4.4. *Let E be a set of ground equations. Let $R \subseteq E$ such that R is convergent and $R \equiv E$. Then E is convergent.*

PROOF. Every equation implied by E has a rewrite proof in R because $R \equiv E$. But $R \subseteq E$, so this is also a rewrite proof in E. $\square$

A corollary of the proposition is that if $\mathit{RedVar}(R) \equiv R$ then R is convergent. Note that if R is reduced, then $\mathit{RedVar}(R) = R$. We will show that L_∞ is equivalent to $\mathit{RedVar}(L_\infty)$, so we can use it in the model construction to show that all true equations are eventually added to the model. But, in order to do that, we must show that $\mathit{RedVar}(L_\infty) \models L_0$.

LEMMA 4.2. *Let $S = S_\infty$ for some fair theorem proving derivation from which the sequences $L_0, \ldots$ and $M_0, \ldots$ have been constructed. Let $R = \mathit{RedVar}(L_\infty)$. Then $R \models L_0$.*

PROOF. We will prove that if $B \in L_0$ then there exists a $B' \in R$ such that $B' \equiv_R B$. Then $R \models B$. So $R \models L_0$.

Suppose that $\Rightarrow B''\theta$ produces B in L_0 and is an instance of the clause $\Rightarrow B'' \in S$. Let $\theta' = \theta\downarrow_R$. Therefore, $B''\theta'$ is substitution reduced by R and $B''\theta' \equiv_R B$.

Let $C = (\Gamma \Rightarrow B')$ be the smallest clause w.r.t. $\prec_r$ in $Gr(S)$ such that $\Gamma \subseteq M_\infty$, B' is substitution reduced by R, and $B' \equiv_R B$. We have just shown there is at least one such B'. C cannot be redundant, otherwise there would be a smaller clause with the desired properties. If B' is not selected, then by Lemma 4.1, there is a smaller clause with the desired properties. We must prove that B' is left-irreducible by R.

So suppose that B' is selected in C and left-reducible by R. If B' is an identity, then $R \models B'$ Otherwise, B' is of the form $u[s] \approx v$, where $u[s] \succ_r v$ and there is an equation $s \approx t$ in R produced by $\Delta \Rightarrow s \approx t \in Gr(S)$, such that $s \approx t$ is substitution reduced by R. So there is a ground instance of a superposition inference in S:

$$\frac{\Delta \Rightarrow s \approx t \qquad \Gamma \Rightarrow u[s] \approx v}{\Delta, \Gamma \Rightarrow u[t] \approx v}.$$

The equation $u[t] \approx v$ must be substitution reduced by R, because $u[s] \approx v$ and $s \approx t$ are substitution reduced by R. By definition of $\prec_r$ extended to clauses, $(\Delta, \Gamma \Rightarrow u[t] \approx v) \prec_r (\Gamma \Rightarrow u[s] \approx v)$, because $(u[t] \approx v) \prec_r (u[s] \approx v)$. All members of Δ and Γ are in M_∞. Finally, $(u[t] \approx v) \equiv_R (u[s] \approx v) \equiv_R B$. Again, we have a smaller clause than C with the desired properties. If it was removed because of redundancy by $C_1, \ldots, C_n$, then one of the C_i must have the desired properties. Therefore, B' must be left-irreducible. This proves that $B' \in R$ and $B' \equiv_R B$. $\square$

Now we show that L_∞ is convergent by showing that it is equivalent to its reduced version. We do that by showing that every member of L_∞ can be reduced to a member in $\mathit{RedVar}(L_\infty)$, by rewriting only with members of $\mathit{RedVar}(L_\infty)$. Proving that L_∞ is convergent also implies that M_∞ is convergent, because $L_\infty \subseteq M_\infty$ and $L_\infty \models M_\infty$ by definition.

LEMMA 4.3. *Let $S = S_\infty$ for some fair theorem proving derivation from which the sequences $L_0, \ldots$ and $M_0, \ldots$ have been constructed. Let $R = \mathit{RedVar}(L_\infty)$. Then $R \equiv L_\infty$.*

PROOF. Let $R = \mathit{RedVar}(L_\infty)$. Obviously, $L_\infty \models R$, since $R \subseteq L_\infty$. We want to prove that $R \models L_\infty$. We will prove that if $B \in L_\infty$ then there exists a $B' \in R$ such that $B' \equiv_R B$. Then $R \models B$. So $R \models L_\infty$ and L_∞ is convergent by Proposition 4.4. The proof is by induction on the size of B w.r.t. $\prec_l$.

Suppose that $\Lambda'\theta \Rightarrow B''\theta$ produces B in L_n and is an instance of the clause $\Lambda' \Rightarrow B'' \in S$. Let $\theta' = \theta\downarrow_R$. We are allowed to do this, because we are assuming that the substitutions in the program clauses have been instantiated. By induction, $R \models \Lambda'\theta'$, and $\Lambda'\theta'$ is substitution reduced by R, so $\Lambda'\theta' \subseteq M_\infty$ because R is convergent by definition and $R \models L_0$ by Lemma 4.2. Also, $B''\theta'$ is substitution reduced by R and $B''\theta' \equiv_R B$.

Let $C = (\Gamma \Rightarrow B')$ be the smallest clause w.r.t. $\prec_r$ in $Gr(S)$ such that $\Gamma \subseteq M_\infty$, B' is substitution reduced by R, and $B' \equiv_R B$. We have just shown there is at least one such B'. C cannot be redundant, otherwise there would be a smaller clause with the desired properties. If B' is not selected, then by Lemma 4.1, there is a smaller clause with the desired properties. We must prove that B' is left-irreducible by R.

So suppose that B' is selected in C and left-reducible by R. If B' is an identity, then $R \models B'$. Otherwise, B' is of the form $u[s] \approx v$, where $u[s] \succ_r v$ and there is an equation $s \approx t$ in R produced by $\Delta \Rightarrow s \approx t \in Gr(S)$, such that $s \approx t$ is substitution reduced by R. So there is a ground instance of a superposition inference in S:

$$\frac{\Delta \Rightarrow s \approx t \qquad \Gamma \Rightarrow u[s] \approx v}{\Delta, \Gamma \Rightarrow u[t] \approx v}.$$

The equation $u[t] \approx v$ must be substitution reduced by R, because $u[s] \approx v$ and $s \approx t$ are substitution reduced by R. By definition of $\prec_r$ extended to clauses, $(\Delta, \Gamma \Rightarrow u[t] \approx v) \prec_r (\Gamma \Rightarrow u[s] \approx v)$, because $(u[t] \approx v) \prec_r (u[s] \approx v)$. All members of Δ and Γ are in M_∞. Finally, $(u[t] \approx v) \equiv_R (u[s] \approx v) \equiv_R B$. Again, we have a smaller clause than C with the desired properties. If it was removed because of redundancy by $C_1, \ldots, C_n$, then one of the C_i must have the desired properties. Therefore, B' must be left-irreducible. This proves that $B' \in R$ and $B' \equiv_R B$. $\square$

It is clear from our definition that every equation appearing in M_∞ is a consequence of program clauses in S, our initial set of clauses. We need to show that all the program clauses are implied by M_∞.

LEMMA 4.4. *Let S_0 be a set of clauses, such that the sequences $L_0, \ldots$ and $M_0, \ldots$ are constructed from S_∞. Then M_∞ implies all the program clauses in $Gr(S_\infty)$.*

PROOF. Let $C = (\Gamma \Rightarrow B) \in Gr(S_0)$, such that $\Gamma' \Rightarrow B' \in S_0$, $\Gamma'\theta = \Gamma$ and $B'\theta = B$. Suppose that $M_\infty \models \Gamma$. We will show that $M_\infty \models B$. Let $\theta' = \theta\downarrow_{M_\infty}$. Then $M_\infty \models \Gamma'\theta'$ and $\Gamma'\theta'$ is substitution reduced by M_∞. So $B'\theta' \in M_\infty$. This is by Lemma 4.3, because $M_\infty \equiv \mathit{RedVar}(L_\infty)$. Therefore $M_\infty \models B$. We have shown that M_∞ implies all program clauses in $Gr(S_0)$. Therefore M_∞ implies all program clauses in $Gr(S_\infty)$. $\square$

Finally, we can show that every true reduced goal reduces to an empty clause whose substitution is a correct answer substitution. This will imply the completeness because initially all clauses have the identity substitution, which is reduced.

THEOREM 4.1. *Let $S_0, \ldots$ be a theorem proving derivation such that the sequences $L_0, \ldots$ and $M_0, \ldots$ have been constructed from S_∞. Let $C = (\Gamma \Rightarrow) [\![\theta]\!] \in Gr(S_\infty)$ such that $\Gamma\theta \subseteq M_\infty$. Let V_G be the set of goal variables. Then there exists a $\theta' = \theta|_{V_G}$ such that $\square [\![\theta']\!] \in Gr(S_\infty)$. Furthermore, if $(\Lambda \Rightarrow) [\![id]\!] \in S_0$ and $M_\infty \models \Lambda\sigma$, then $\square [\![\sigma']\!] \in Gr(S_\infty)$, for some $\sigma' = \sigma \downarrow_{M_\infty} |_{V_G}$.*

PROOF. Let $(\Gamma' \Rightarrow) [\![\theta']\!]$ be the smallest clause (w.r.t. $\prec_l$) such that $\theta' = \theta|_{V_G}$ and $M_\infty \models \Gamma'\theta'$. By Lemma 4.1, if $(\Gamma' \Rightarrow) [\![\theta']\!]$ has a negative literal selected, then there is a smaller clause with the required properties. Therefore, $(\Gamma' \Rightarrow) [\![\theta']\!]$ must be the empty clause.

If $(\Lambda \Rightarrow) [\![id]\!] \in S_0$. and $M_\infty \models \Lambda\sigma$, then $\Lambda\sigma' \in M_\infty$, therefore $\square [\![\sigma']\!] \in I(S_\infty)$. $\square$

This shows the completeness of the inference system with respect to answer substitutions. The refutational completeness follows from that.

5. Deletion Rules

In this section we explain which deletion rules preserve completeness. We write deletion rules in the form $S \vdash T$ to indicate that a set of clauses S may be replaced by a set of clauses T. We say that a deletion rule $S \vdash T$ is *correct* if for each $C \in T$, $S \models C$ and for each $C \in S \setminus T$, C is redundant in S.

TAUTOLOGY DELETION

$$S \cup \{\Gamma \Rightarrow t \approx t\} \vdash S.$$

PROPOSITION 5.1. *Tautology Deletion is a correct deletion rule.*

PROOF. If C is a tautology of the form $\Gamma \Rightarrow t \approx t$ then C is redundant in any set S of clauses. $\square$

The form of tautology deletion which removes a clause of the form $A, \Gamma \Rightarrow A$ does not preserve completeness. The following example illustrates why such tautology deletions are not allowed. For this example we assume an ordering such that $b \succ_r c$. For all the examples in this section, we assume a selection rule such that the positive literal is selected in each program clause.

Suppose we have:

1. $\Rightarrow P(c, b, b)$
2. $P(c, c, b), P(c, b, c) \Rightarrow b \approx c$
3. $P(x, y, y) \Rightarrow P(x, y, x)$
4. $P(x, y, y) \Rightarrow P(x, x, y)$
5. $P(c, c, c) \Rightarrow$.

The conclusion of every inference in this set of clauses is identical to an existing clause or is a tautology. However, the set of clauses is unsatisfiable. If we keep the tautologies, we can generate the empty clause, as we show below. This example shows that tautologies of the form $A, \Gamma \Rightarrow A$ may not be deleted. However, we may disallow goal paramodulation and resolution inferences which have such a tautology as a premise. That is because such a clause is not productive.

Now we show how the empty clause is generated if tautologies are kept. Tautologies $P(c,c,b), P(c,b,c) \Rightarrow P(c,c,b)$ and $P(c,c,b), P(c,b,c) \Rightarrow P(c,b,c)$ are both created by inferences between clauses 1 and 2. Then, an inference of either of these two clauses with clause 2 gives us $P(c,c,b), P(c,b,c) \Rightarrow P(c,c,c)$. This clause can then be used in an inference with $P(c,c,c) \Rightarrow$ to generate $P(c,c,b), P(c,b,c) \Rightarrow$. Next, an inference with clause 4 into $P(c,c,b)$ gives us $P(x,y,y), P(c,b,c) \Rightarrow [\![x = c \wedge y = b]\!]$. An inference with clause 1 removes the first subgoal to give us $P(c,b,c) \Rightarrow$. Then we apply clause 3 to this subgoal. The result is $P(x,y,y) \Rightarrow [\![x = c \wedge y = b]\!]$. Resolving this with clause 1 gives us the empty clause.

Subsumption can also be defined. However, we are only allowed to remove a clause $D[\![\theta]\!]$ if some clause $C[\![\sigma]\!]$ exists such that $C \subset D$ and $C\sigma \subseteq D\theta$, as opposed to the usual form of subsumption where we can remove it if $C\sigma \subseteq D\theta$.

SUBSUMPTION

$$S \cup \{C[\![\sigma]\!], D[\![\theta]\!]\} \vdash S \cup \{C[\![\sigma]\!]\}$$

if $C \subseteq D$ and $C\sigma \subseteq D\theta$.

PROPOSITION 5.2. *Subsumption is a correct deletion rule.*

PROOF. If $C[\![\sigma]\!]$ and $D[\![\theta]\!]$ are both program clauses or both goal clauses, $C \subseteq D$ and $C\sigma \subseteq D\theta$, then $D[\![\theta]\!]$ is redundant w.r.t. $C[\![\sigma]\!]$. □

If C and D are program clauses, then we could allow the deletion if some renaming of C is a subset of D. The general form of subsumption, which allows the removal if $C\sigma \subseteq D\theta$ is not complete, as we illustrate with the following example. In the example, $a \succ_r b$.

Suppose we have:

1. $\Rightarrow P(a,a)$
2. $P(a,a) \Rightarrow a \approx b$
3. $P(x,x) \Rightarrow P(x,b)$
4. $P(x,x) \Rightarrow P(b,x)$
5. $P(b,b) \Rightarrow$

Every inference among this set of clauses results in a clause which is identical to an existing clause or is subsumed by an existing clause. However, the set is unsatisfiable. If the subsumed clauses are kept, then the empty clause is generated, as we show below. So we cannot allow the deletion of a subsumed clause in general. However, we may disallow a goal paramodulation or resolution inference if the program clause C in the inference is subsumed, because C would not be productive.

We now show how to generate the empty clause if subsumed clauses are kept. Inferences among clauses 1 and 2 generate $P(a,a) \Rightarrow P(a,b)$ and $P(a,a) \Rightarrow P(b,a)$. These clauses are subsumed by clauses 3 and 4. If the clauses are kept, an inference between either of these clauses and clause 2 gives us $P(a, a) \Rightarrow P(b, b)$. Now it is easy to generate the empty clause. An inference between this clause and $P(b, b) \Rightarrow$ gives $P(a, a) \Rightarrow$, which is immediately refuted by $\Rightarrow P(a, a)$.

We have defined subsumption so that a goal clause cannot subsume a program clause. This is to preserve the property of answer substitutions. For instance, an empty clause could then subsume everything, but it would only represent one answer substitution.

All simplifications are allowed.

SIMPLIFICATION

$$S \cup \{C[s\sigma] [\![\theta]\!], \Rightarrow s \approx t\} \vdash S \cup \{C[t\sigma] [\![\theta]\!], \Rightarrow s \approx t\}.$$

We could also present a blocking rule, as in Bachmair *et al.* (1995), to say that clauses with reducible substitutions may be deleted.

The proof that simplified clauses may be deleted is not as simple as the proof of the other deletion rules. We must prove that the deleted clause is redundant.

PROPOSITION 5.3. *Simplification is a correct deletion rule.*

PROOF. We need to show that $C[s\sigma] [\![\theta]\!]$ is redundant in any set of clauses containing $C[t\sigma] [\![\theta]\!]$ and $s \approx t$. Conditions 1 and 2 are satisfied by definition. But it is necessary to prove that $C[t\sigma] [\![\theta]\!] \prec_l C[s\sigma] [\![\theta]\!]$.

For each instance C' of $C[s\sigma] [\![\theta]\!]$, there is a corresponding instance D of $C[t\sigma] [\![\theta]\!]$ which implies C' in combination with the corresponding instance $s' \approx t'$ of $s \approx t$. Suppose that $level(C') = n$ where n is a non-negative integer or $n = \infty$. Obviously, $D \prec_r C'$ and $s' \approx t' \prec_r C'$. Since $s' \approx t'$ is a fact, we know that $level(s' \approx t') = 0$. Therefore, $s' \approx t' \prec_l C'$. In order to show that $D \prec_l C'$, we must show that $level(D) \leq n$.

Since $level(C') = n$, there is a convergent set of equations $R \subseteq L_n$ such that $R \models L_0$, $R \models C'$ and C' is substitution reduced by R. Since C' is substitution reduced by R, D also must be substitution reduced by R.

The equation $s' \approx t'$ might not be substitution reduced by R, but let A be the result of reducing the variables of $s' \approx t'$ by R. Then $s' \approx t'$ is implied by $R \cup A$. This implies that all equations in A are at a level less than or equal to n. Since all equations of R are also at a level less than or equal to n, and $R \cup A \models D$, we know that $level(D) \leq n$. Therefore $D \prec_l C'$. □

6. Conditional Narrowing

The result of this paper can be applied to *Conditional Narrowing. Basic Conditional Narrowing* can be viewed as the restriction of our inference system to the Goal Paramodulation and Equation Resolution inference rule. It is called Conditional Narrowing if we allow paramodulation into substitution positions, but no other variable positions. Conditional Narrowing is not complete in general for equational Horn clauses, but we have proved that it is complete if the set of program clauses is saturated by Superposition[†].

Now we compare our result with related results on Conditional Narrowing. First we

[†] We are specifically referring to the selection rule which selects positives in program clauses, although this is true for any selection rule.

need some notation. Let $Var(t)$ be the set of variables in t, for any object t. Let $C = (\Gamma \Rightarrow s \approx t)$ be a program clause. C is type 1 if $Var(\Gamma) \cup Var(t) \subseteq Var(s)$, type 2 if $Var(t) \subseteq Var(s)$, and type 3 if $Var(t) \subseteq Var(s) \cup Var(\Gamma)$. Let S be a set of clauses. For all i, S is of type i if C is of type i for all $C \in S$. Let R_0 be the set of equations appearing as facts in S and let R_n be the set of equations A such that there is an equation $\Gamma \Rightarrow A$ in $Gr(S)$ such that $R_{n-1} \models \Gamma$. Let $R_\infty = \bigcup_{n \geq 0} R_n$. S is *convergent* if R_∞ is convergent. S is *level-convergent* if each R_n is convergent. We compare this with the similar sets constructed in Definition 3.1. It is evident that $M_n \subseteq R_n$ for all n. We have proved that, if the set of clauses is saturated by Superposition, then $M_\infty \equiv R_\infty$.

Conditional narrowing is complete for convergent sets of type 1 (Kaplan, 1984), level-convergent sets of type 2 (Giovanetti and Moiso, 1986), and level-convergent sets of type 3 (Middledorp and Hamoen, 1994). It is not complete for convergent sets of type 2 (Giovanetti and Moiso, 1986). Basic Conditional Narrowing is complete for level-convergent sets of type 2 (Giovanetti and Moiso, 1986), but not complete for convergent sets of type 1 (Middledorp and Hamoen, 1994) (see Middledorp and Hamoen (1994) for a good exposition of all these results). In this paper we showed that Basic Conditional Narrowing is complete for sets which are saturated by Superposition. A big advantage of our result is that we require no restrictions on where variables appear, as needed in all previous results. A set saturated by Superposition is convergent but not necessarily level-convergent. For instance, consider the example given earlier in the paper:

1. $\Rightarrow P(a, b)$
2. $P(x, b) \Rightarrow a \approx c$
3. $Q(a, b) \Rightarrow a \approx b$
4. $Q(a, b) \Rightarrow P(b, b)$

This is saturated by superposition. However, suppose we construct each R_i. Then $R_0 = \{P(a, b)\}$, $R_1 = \{P(a, b), Q(a, b)\}$, $R_2 = \{P(a, b), Q(a, b), a \approx b, P(b, b)\}$, and $R_4 = \{P(a, b), Q(a, b), a \approx b, P(b, b), Q(b, b)\}$. Note that R_3 is not convergent, since it contains $Q(a, b)$ and $a \approx b$, but not $Q(b, b)$.

The condition of being saturated by Superposition is a natural condition. It seems to be the natural way to test the other convergence conditions anyway. Another advantage of our method is that there is an algorithm to create the condition for completeness of Basic Conditional Narrowing: saturate the set by Superposition. This is also the case for the methods of Dershowitz (1991), Nieuwenhuis and Nivela (1991); Bachmair and Ganzinger (1994); Bachmair *et al.* (1995); Kounalis and Rusinowitch (1988); Rusinowitch (1988); Hsiang and Rusinowitch (1991); Pais and Peterson (1991); Bachmair and Ganzinger (1990), which are special cases of our general framework.

The following example from Middledorp and Hamoen (1994) illustrates why a set saturated by Superposition is different from a convergent set:

1. $\Rightarrow a \approx b$
2. $\Rightarrow a \approx c$
3. $x \approx b, x \approx c \Rightarrow b \approx c$
4. $b \approx c \Rightarrow$

This set is convergent but, as Middledorp and Hamoen (1994) show, Conditional Narrowing is not complete for this set. However, our inference rules would force an inference between the first two clauses to create the clause $\Rightarrow b \approx c$. Then it is saturated by Superposition, and Conditional Narrowing is complete.

7. Conclusion

In this paper, we have given a new general framework to prove completeness results for theorem proving and logic programming with horn clauses with equality. The usual concept of a selection rule is extended to program clauses. Then we use the selection rule to define a schema of inference rules. The schema encompasses previously known complete inference systems for Horn clauses, as we detail in Section 3. Additionally, we show that the schema includes an inference system conjectured complete by Fribourg, and also covers some new inference systems. The system of Fribourg is especially interesting, because heads of program clauses cannot be used in an inference with a condition of another clause. Thus the inference system is a combination of a bottom up completion process and a top down goal solving process. The completeness proof of our inference rules is especially interesting, because it was necessary to develop new techniques, since the old techniques only work for specific instances of the schema. Our results are also interesting because of the connection with previous work on Conditional Narrowing. In Section 6, we detail the relationship between our work and previous work in this area.

We can use the idea of selection rule to relate completeness results on Horn clauses with equality to completeness results on general clauses with equality. Previously, every selection rule complete for Horn clauses with equality was also a complete selection rule for general clauses with equality. We extend the notion of selection rules to map to sets of literals, so that for a selection rule Sel and a clause C, $Sel(C)$ is a non-empty subset of C. Then any inference with C must involve one of the literals in this set. A literals L is maximal in a clause C if L is in C and there is no L' in C such that $L' > L$. Let $\mathcal{F}$ be the set of all selection rules such that if $Sel \in \mathcal{F}$ and C is a clause, then either

1. $Sel(C)$ is a set consisting of one negative literal in C, or
2. $Sel(C)$ is the set of all maximal literals in C.

For general clauses, the best result on selection rules is from Bachmair *et al.* (1995). That inference system is complete when using a selection rule in $\mathcal{F}$. However, it is not known to be complete for any selection rule outside of $\mathcal{F}$. This was previously also the best known result for Horn clauses with equality. In this paper, we have improved the result for Horn clauses. In the above notation, we have shown that Basic Superposition for Horn Clauses is complete for every selection rule, since completeness for one literal selected in each clause implies completeness for more than one literal selected. This is an improvement, because the selection rule that selects a positive literal in each program clause is not included in $\mathcal{F}$.

However, for some of the selection rules proved complete in this paper, there is no natural extension to first order logic with equality that is complete. For instance, consider the selection rule that selects a positive literal for each clause containing one, otherwise selects a negative literal (as in SLD resolution). That rule would be useful in the first order case, because it would give a selection rule for paramodulation that is somewhat goal directed but still uses rewrite techniques. Unfortunately, we have a counterexample which shows that even the most general extension of this selection rule to first order logic is not complete. Consider a selection rule on general clauses which selects all positive literals in a clause which contains positive literals and some negative literal otherwise. Consider the following set of clauses, where the selected literals are underlined:

1. $\neg p \vee \underline{r}$
2. $\neg q \vee \underline{p}$
3. $\neg r \vee \underline{q}$
4. $\neg p \vee \underline{\neg q}$
5. $\neg q \vee \underline{\neg r}$
6. $\neg r \vee \underline{\neg p}$
7. $\underline{p} \vee \underline{q} \vee \underline{r}$.

This set is unsatisfiable, but we cannot generate the empty clause, even if we keep tautologies. The reader will notice that the conclusion of every inference is subsumed by the original set, so the empty clause will never be generated. Since the extension of this strategy to a Resolution-based inference system is not complete, we are now considering an extension of this strategy to a Model Elimination inference system.

In this paper, we have given a method of applying Basic Paramodulation to Horn clauses with equality. All previous results are bottom-up, data-driven inference systems, or else they do not admit ordered completion and simplification techniques. We give a method of combining a bottom-up inference system with a top-down, goal-directed inference system (like SLD resolution). We believe the goal-directedness of our strategy will make it perform better in practice. We have presented the inference system in an abstract setting which should make it possible to combine it with other theorem proving strategies, like the redex restrictions from Bachmair *et al.* (1995). One practical problem with our method (as in SLD-resolution) is that the goal-directedness might blow up the search space. However, in a recent result (Lynch, 1995), we give a practical method to prevent this blowup and to restrict the search space drastically.

Acknowledgements

Thanks go to Michaël Rusinowitch, Wayne Snyder, and Claude Kirchner for helpful discussions, and Leo Bachmair for pointing out an error in one of the definitions.

References

Bachmair, L. (1991). *Canonical Equational Proofs.* Boston, MA: Birkhauser Boston, Inc.

Bachmair, L., Dershowitz, N., Plaisted, D. (1989). Completion without failure. *Resolution of Equations in Algebraic Structures* **2**, pp. 1–30.

Bachmair, L., Ganzinger, H. (1990). On restrictions of ordered paramodulation with simplification. In *Proc. 10th Int. Conf. on Automated Deduction*, LNCS, vol. 449, pp. 427–441, Berlin: Springer-Verlag.

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

Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W. (1992). Basic paramodulation. In *Proc. 11th Int. Conf. on Automated Deduction*, Lect. Notes in Artificial Intelligence **607**, pp. 462–476, Berlin: Springer-Verlag.

Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W. (1995). Basic paramodulation. *Information and Computation* **121**(2):172–192.

Brand, D. (1975). Proving theorems with the modification method. *SIAM J. Computing* **4**(4):412–430.

Dershowitz, N. (1991). A Maximal Literal Unit Strategy for Horn Clauses. *Proc. 2nd Int. Workshop on Conditional Term Rewriting Systems*, LNCS, vol. 516, pp. 14–25.

Dershowitz, N., Jouannaud, J.-P. (1990). Rewrite Systems. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, pp. 243–320. Amsterdam: North-Holland.

Fribourg, L. (1984). Oriented equational clauses as a programming language. *J. Logic Programming* **1**(2):165–177.

Giovanetti, E., Moiso, C. (1986). A Completeness Result for E-Unification Algorithms based on Conditional Narrowing. In *Proc. of the Workshop on Foundations of Logic and Functional Programming*, LNCS, vol. 306, pp. 157–167

Hullot, J.-M. (1980). Canonical forms and unification. In *Proc. 5th Int. Conf. on Automated Deduction*, LNCS, vol. 87, pp. 318–334, Berlin: Springer-Verlag.

Hsiang, J., Rusinowitch, M. (1991). Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *J. Association for Computing Machinery* **38** pp. 559–587.

Kaplan, S. (1984). Conditional rewrite rules. *Theoretical Computer Science* **33**(2):175–193.

Kirchner, C., Kirchner, H., Rusinowitch, M. (1990). Deduction with symbolic constraints. *Revue Francaise d'Intelligence Artificielle* **4**(3):9–52.

Kounalis, E., Rusinowitch, M. (1988). On Word Problems in Horn Logic. *Proc. 1st Int. Workshop on Conditional Term Rewriting Systems*, LNCS, vol. 308, pp. 144–160.

Lynch, C. (1995). Paramodulation without duplication. In *Proc. 10th Int. IEEE Symp. on Logic in Computer Science*, San Diego.

Middledorp, A., Hamoen, E. (1994). Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing* **5**:213–253.

Nieuwenhuis, R. (1995). On Narrowing, Refutation Proofs and Constraints. In *Proc. 6th Int. Conf. on Rewriting Techniques and Applications*, LNCS, vol. 914, pp. 56–70. Berlin, Springer-Verlag.

Nieuwenhuis, R., Nivela, P. (1991). Efficient deduction in Equality Horn Logic by Horn-completion. *Information Processing Letters* **39** pp. 1–6.

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

Nieuwenhuis, R., Rubio, A. (1995). Theorem proving with ordering and equality constrained clauses. *J. Symbolic Computation* **19**(4):321–352.

Pais, J., Peterson, G. (1991). Using forcing to prove completeness of resolution and paramodulation. *J. Symbolic Computation* **11**:3–19.

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

Rusinowitch, M. (1988). Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure to a complete set of inference rules. In *Proc. International Conference on Fifth Generation Computer Systems.*

Snyder, W., Lynch, C. (1991a). An Inference System for Horn Clause Logic with Equality. *Proc. 2nd Int. Workshop on Conditional Term Rewriting Systems*, LNCS, vol. 516, pp. 454–461.

Snyder, W., Lynch, C. (1991). Goal directed strategies for paramodulation. In *Proc. 4th Int. Conf. on Rewriting Techniques and Applications*, LNCS, vol. 488, pp. 150–161, Berlin: Springer-Verlag.