PROVING FIRST-ORDER EQUALITY THEOREMS WITH HYPER-LINKING

by

Geoffrey David Alexander

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill, 1995

Approved by:

______________________________ Advisor

______________________________ Reader

______________________________ Reader

GEOFFREY DAVID ALEXANDER

Proving First-Order Equality Theorems with Hyper-Linking

(Under the direction of Professor David A. Plaisted)

ABSTRACT

Lee and Plaisted recently developed a new automated theorem proving strategy called *hyper-linking*. As part of his dissertation, Lee developed a round-by-round implementation of the hyper-linking strategy, which competes well with other automated theorem provers on a wide range of theorem proving problems. However, Lee's round-by-round implementation of hyper-linking is not particularly well suited for the addition of special methods in support of equality.

In this dissertation, we describe, as alternative to the round-by-round hyper-linking implementation of Lee, a *smallest instance first* implementation of hyper-linking which addresses many of the inefficiencies of round-by-round hyper-linking encountered when adding special methods in support of equality. Smallest instance first hyper-linking is based on the formalization of generating smallest clauses first, a heuristic widely used in other automated theorem provers. We prove both the soundness and logical completeness of smallest instance first hyper-linking and show that it always generates smallest clauses first under a large class of ordering on clauses. We discuss which of the refinements used by Lee in round-by-round hyper-linking are also applicable to smallest instance first hyper-linking.

We add equality support to smallest instance first hyper-linking using unfailing completion, narrowing, and Brand's equality transformation in a method we call *unfailing completion equality support*. We show the soundness and logical completeness of smallest instance first hyper-linking with unfailing completion equality support. We also show that Brand's equality transformation is unnecessary for the case where all positive equational literals occur in equational Horn clauses. In particular, this means that for pure equational problems, smallest instance first hyper-linking with unfailing completion equality support simply reduces to an application of unfailing completion.

We conclude the dissertation by describing our preliminary implementation of smallest instance first hyper-linking. We present a comparison of our implementation with that of Lee's on the 2,655 problems in the TPTP Problem Library of Sutcliffe and Suttner.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. David A. Plaisted, for his guidance and support throughout the duration of my dissertation research. This dissertation would have never been completed without his kind assistance and advice.

I would like to thank the readers of my dissertation, Dr. Don Stanat and Dr. Gyula A. Magó, for their numerous suggestions for improving the style and presentation of my dissertation. I would also like to thank my other committee members, Dr. Michael Resnik and Dr. James H. Anderson.

I would like to thank fellow students Shie-Jue Lee for his assistance in helping me understand the workings of CLIN and Heng Chu for the numerous conversations and discussions we had. I would also like to thank fellow students George Greene, Muthukrishnan Paramasivam, Bill Yakowenko, and Yushan Zhu for their support.

I would like to thank IBM for its financial support during my research.

Finally, I would like to thank my parents for their love and support over the years, to whom I dedicate this dissertation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Lee and Plaisted[63] have recently developed a new instance based theorem proving strategy called hyper-linking. Lee[62] showed that the hyper-linking strategy competes well with the much studied resolution theorem proving strategy. Part of Lee's work included the development of round-by-round hyper-linking, which he implemented in the automated theorem prover CLIN. However, Lee's implementation did not include special methods for proving theorems involving equality. For this reason, CLIN does not compete well on equality theorems with automated theorem provers having specialized equality support, such as McCune's OTTER[70]. For hyper-linking to become a viable general-purpose theorem proving strategy, it must be shown that the hyper-linking strategy can efficiently prove equality theorems.

While investigating methods for adding equality support to hyper-linking, we encountered a number of inefficiencies in round-by-round hyper-linking, which prove particularly troublesome when adding methods in support of equality. To eliminate these inefficiencies, we have developed an alternative hyper-linking implementation which we refer to as smallest instance first hyper-linking. We add equality support to smallest instance first hyper-linking through the use of unfailing completion in a method we refer to as unfailing completion equality support.

It is our belief that

> *Smallest instance first hyper-linking is a viable alternative to the round-by-round hyper-linking implementation of Lee. In particular, smallest*

instance first hyper-linking is well suited for the addition of special methods in support of equality such as unfailing completion equality support.

We begin by presenting background material on first-order logic and automated theorem proving in Chapter 2. In Chapter 3, we describe our attempt to add equality support to round-by-round hyper-linking. We discuss a number of inefficiencies encountered. In Chapter 4, we present smallest instance first hyper-linking and show that it is sound and logically complete. We also discuss a number of enhancements to smallest instance first hyper-linking. In Chapter 5, we offer unfailing completion equality support as a method for adding equality support to smallest instance first hyper-linking. In Chapter 6, we describe our preliminary implementation of smallest instance first hyper-linking, CLIN-2. We include results of running both CLIN and CLIN-2 on the 2,655 theorem proving problems in version 1.1.3 of the TPTP (Thousands of Problems for Theorem Provers) Problem Library of Sutcliffe and Suttner[93,94]. In Chapter 7, we give our conclusions and present directions for future work.

Chapter 2

Background

In this chapter we present background material on mathematical logic, automated theorem proving, term rewriting, and hyper-linking. The purpose of this chapter is to introduce the notation used in the remainder of the dissertation. While this chapter serves as a reference to aid in the reading of the following chapters, it is not intended as a tutorial. For those readers requiring further information, we suggest Boolos and Jeffrey[11], Ebbinghaus, Flum, and Thomas[32], and Mendelson[71] for mathematical logic, Chang and Lee[17], Fitting[35], Gallier[37], and Loveland[66] for automated theorem proving, Bachmair, Dershowitz, and Plaisted[5], Dershowitz[25,26,27,28], Dershowitz and Jouannaud[29], Knuth and Bendix[58], and Plaisted[82] for term rewriting, and Lee[62] and Lee and Plaisted[63] for hyper-linking. To aid the reader, we have identified with an asterisk (*) those theorems which are directly referenced in later chapters.

2.1 First-Order Syntax

We begin by defining an alphabet for first-order logic.

Definition 2.1 An *alphabet* A is a set of symbols divided into the following six classes:

Punctuation:	,	(*comma*)
	(	(*open parenthesis*)
	)	(*close parenthesis*)
Truth Symbols:	**T**	(*true*)
	F	(*false*)

Connectives:	$\wedge$ *(and)*
	$\vee$ *(or)*
	$\sim$ *(not)*
	$\rightarrow$ *(implication)*
	$\equiv$ *(equivalence)*
Quantifiers:	$\forall$ *(universal quantifier)*
	$\exists$ *(existential quantifier)*
Constants:	$f_i^n,\ i \geq 1, n \geq 0$ *(function symbols)*
	$p_i^n,\ i \geq 1, n \geq 0$ *(predicate symbols)*
Variables:	$x_i,\ i \geq 1$ *(variables)*

We refer to n as the *arity* of the function symbol f_i^n or predicate symbol p_i^n, the function symbols f_i^0 as *constants*, and the predicate symbols p_i^0 as *propositional constants*. ∎

For all alphabets, the punctuation, truth symbols, connectives, and quantifiers are fixed. Only the constants and variables vary between different alphabets. One should take special notice that an alphabet can contain at most a countably infinite number of constants and variables.

The definition of an alphabet given above is, by no means, the definitive definition. There are various other definitions. For example, Mendelson[71] defines an alphabet with $\sim$ and $\rightarrow$ as the only connectives and $\forall$ as the only quantifier. He defines the connectives $\wedge$, $\vee$, and $\equiv$ and the quantifier $\exists$ informally in terms of $\sim$, $\rightarrow$, and $\forall$.

We now define a first-order language defined by an alphabet.

Definition 2.2 Let A be an alphabet. A *term* is any string on A defined inductively as follows:

(a) Every constant f_i^0 in A is a term.
(b) Every variable x_i in A is a term.
(c) If f_i^n is a function symbol in A and $t_1, \ldots, t_n$ are terms with $n \geq 1$, then $f_i^n(t_1, \ldots, t_n)$ is a term.

A term is *ground* if it contains no variables. ∎

Definition 2.3 Let A be an alphabet. An *atom* is any string on A defined as follows:

(a) The truth symbols **T** and **F** are atoms.
(b) Every propositional constant p_i^0 in A is an atom.
(c) If p_i^n is a predicate symbol in A and $t_1, \ldots, t_n$ are terms with $n \geq 1$, then $p_i^n(t_1, \ldots, t_n)$ is an atom.

An atom is *ground* if it contains no variables. ∎

Definition 2.4 Let A be an alphabet. A *formula* is any string on A defined inductively as follows:

(a) Every atom is a formula.
(b) If F and G are formulas, then $(F \wedge G)$ is a formula.
(c) If F and G are formulas, then $(F \vee G)$ is a formula.
(d) If F is a formula, then $({\sim}F)$ is a formula.
(e) If F and G are formulas, then $(F \rightarrow G)$ is a formula.
(f) If F and G are formulas, then $(F \equiv G)$ is a formula.
(g) If x_i is a variable and F is a formula, then $((\forall x_i)\, F)$ is a formula.
(h) If x_i is a variable and F is a formula, then $((\exists x_i)\, F)$ is a formula.

A formula of the form $(F \wedge G)$ is the *conjunction* of the *conjuncts* F and G. A formula of the form $(F \vee G)$ is the *disjunction* of the *disjuncts* F and G. A formula of the form $({\sim}F)$ is the *negation* of the formula F. A formula F is the *scope* of a formula of the form $(Qx_i)\, F$ where Q is either $\forall$ or $\exists$. The occurrence of a variable x_i in a formula is *bound* if it is the occurrence of x_i in a quantifier $(\forall x_i)$ or $(\exists x_i)$ or if it lies within the scope of a quantifier $(\forall x_i)$ or $(\exists x_i)$; otherwise it is *free*. A formula is *closed* if it contains no free variables; otherwise, it it is *open*. A formula is *ground* if it contains no variables. A formula is a *propositional formula* if it contains only parentheses, connectives, propositional constants, and truth symbols. ■

Formulas are also referred to as well-formed formulas or wffs in the literature. Example formulas are $\mathbf{T}$, $(\mathbf{F} \wedge (p_1^1(x_i) \vee p_1^0))$, and $((\forall x_1)((\exists x_2)\, p_1^2(f_1^2(x_1, x_2), f_1^0)))$. The formal syntax for formulas given above can become quite cumbersome. We, therefore, use a number of abbreviations to simplify the syntax. We omit superscripts on function symbols and predicate symbols in formulas. When referring to individual function symbols or predicate symbols, we use superscripts only when necessary to avoid ambiguity. We use the symbols a, b, and c, with or without subscripts, for constants. We use special symbols such as 0, 1, and e for constants when appropriate. We use the symbols f, g, and h, with or without subscripts, for function symbols other than the constants. We use the symbols p, q, and r, with or without subscripts, for predicate symbols. We use the symbols v, w x, y, z, with or without subscripts, for variables. We sometimes use infix symbols such as $+$, $\times$, and $=$ for function symbols and predicate symbols to improve readability. Also, special notation such as x^{-1} is sometimes used for function symbols to improve readability.

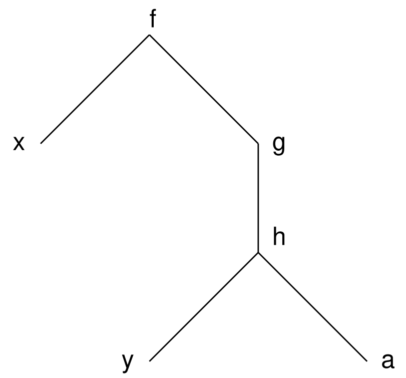

Figure 2.1: Tree Structure for the Term $f(x, g(h(y, a)))$

Parentheses are omitted whenever possible to improve readability. In particular, $(\exists x)$ and $(\forall x)$ are written as $\exists x$ and $\forall x$ respectively. We also use braces $\{\}$ and brackets $[]$ in addition to parentheses to improve clarity. For example, we can abbreviate the formula $((\forall x_1)((\exists x_2)\, p_1^2(f_1^2(x_1, x_2), f_1^0)))$ with $\forall x\, \exists y\, x + y = e$ by eliminating parentheses and using x for x_1, y for x_2, the infix operator $=$ for p_1^2, the infix operator $+$ for f_1^2, and e for f_1^0.

A term can be viewed as a tree. Figure 2.1 shows the tree structure for the term $f(x, g(h(y, a)))$. Each symbol in the term can be identified by its position within the tree. For example, using a decimal notation, we say the position of the symbol h in $f(x, g(h(y, a)))$ is 2.1 and the position of the symbol a in $f(x, g(h(y, a)$ is 2.1.2.

Definition 2.5 Let T be a set of terms and s, t, and u terms in T. We say s is a *subterm* of t if s occurs in t. The *position* p of s in t is the position of s's outer function symbol. We use the notation $t[s]_p$ to indicate that s is a subterm of t at position p and the notation $t[s \rightarrow u]_p$ to denote the term obtained by replacing the subterm s of t at position p by the term u. ∎

An atom can also be viewed as a tree. Thus, each symbol in an atom can be identified by its position in the tree in a manner similar to that for terms.

We define the languages defined by an alphabet.

Definition 2.6 A *propositional language* is the set of all propositional formulas formed by an alphabet. ∎

Definition 2.7 A *first-order language* is the set of all formulas formed by an alphabet. ∎

2.2 First-Order Semantics

In addition to being able to describe a first-order language syntactically, we would also like to be able to discuss the truth or falsity of formulas in the first-order language. We do this by assigning a meaning to each of the symbols in the alphabet of the first-order language. We use the meaning of the symbols to define the meaning of terms. We then use the meaning of terms to define the truth value of atoms and formulas.

Definition 2.8 Let L be a first-order language. An *interpretation* I is a triple $\langle D, I_f, I_p \rangle$ where

(a) D is a non-empty set.
(b) I_f maps each function symbol f^n into a function $(f^n)^I : D^n \rightarrow D$.
(c) I_p maps each predicate symbol p^n into a function $(p^n)^I : D^n \rightarrow \{\mathit{true}, \mathit{false}\}$.

We refer to D as the *domain* of the interpretation, $(f^n)^I$ as the *meaning* of the function symbol f^n in I, and $(p^n)^I$ as the *meaning* of the predicate symbol p^n in I. ∎

I_f maps a constant a into a function $a^I : D^0 \rightarrow D$. This simply means that a^I is an element of D. Similarly, I_p maps a propositional constant p into a function $p^I : D^0 \rightarrow \{\mathit{true}, \mathit{false}\}$. This simply means that p^I is an element of $\{\mathit{true}, \mathit{false}\}$.

Definition 2.9 Let L be a first-order language, I an interpretation, and D the domain of I. A *variable assignment* v maps each variable x into an element $v(x)$ of D. ∎

Definition 2.10 Let L be a first-order language, I an interpretation, v a variable assignment, and t a term. The *meaning* $t^{I,v}$ of t is inductively defined as follows:

(a) If t is a variable x, then $t^{I,v}$ is $v(x)$.
(b) If t is a constant a, then $t^{I,v}$ is a^I.
(c) If t is of the form $f(t_1, \ldots, t_n)$ where each t_i for $1 \leq i \leq n$ is a term, then $t^{I,v}$ is $f^I(t_1^{I,v}, \ldots, t_n^{I,v})$. ∎

Definition 2.11 Let L be a first-order language, I an interpretation, v a variable assignment, and A an atom. The *meaning* $A^{I,v}$ of A is defined as follows:

(a) If A is $\mathbf{T}$, then $A^{I,v}$ is *true*.
(b) If A is $\mathbf{F}$, then $A^{I,v}$ is *false*.

(c) If A is a propositional constant p, then $A^{I,v}$ is p^I.
(d) If A is of the form $p(t_1, \ldots, t_n)$ where each t_i for $1 \le i \le n$ is a term, then A^I is $p^I(t_1^{I,v}, \ldots, t_n^{I,v})$. ∎

Definition 2.12 Let L be a first-order language, I an interpretation, D the domain of I, v a variable assignment, and F an formula. The *meaning* $F^{I,v}$ of F is inductively defined as follows:

(a) If F is an atom G, then $F^{I,v}$ is $G^{I,v}$.
(b) If F is of the form $G \wedge H$ where G and H are formulas, then $F^{I,v}$ is *true* if both $G^{I,v}$ and $H^{I,v}$ are *true*; otherwise, $F^{I,v}$ is *false*.
(c) If F is of the form $G \vee H$ where G and H are formulas, then $F^{I,v}$ is *false* if both $G^{I,v}$ and $H^{I,v}$ are *false*; otherwise, $F^{I,v}$ is *true*.
(d) If F is of the form $\sim G$ where G is a formula, then $F^{I,v}$ is *true* if $G^{I,v}$ is *false*; otherwise, $F^{I,v}$ is *false*.
(e) If F is of the form $G \rightarrow H$ where G and H are formulas, then $F^{I,v}$ is *false* if $G^{I,v}$ is *true* and $H^{I,v}$ is *false*; otherwise, $F^{I,v}$ is *true*.
(f) If F is of the form $G \equiv H$ where G and H are formulas, then $F^{I,v}$ is *true* if $G^{I,v}$ and $H^{I,v}$ have the same value; otherwise, $F^{I,v}$ is *false*.
(g) If F is of the form $\forall x\, G$ where G is a formula, then $F^{I,v}$ is *true* if for all $d \in D$, $G^{I,v'}$ is *true* where v' is v except that v' maps x into d; otherwise, $F^{I,v}$ is *false*.
(h) If F is of the form $\exists x\, G$ where G is a formula, then $F^{I,v}$ is *true* if there exists a $d \in D$, $G^{I,v'}$ is *true* where v' is v except that v' maps x into d; otherwise, $F^{I,v}$ is *false*. ∎

Conjunctions of the forms $(F \wedge G) \wedge H$ and $F \wedge (G \wedge H)$ have the same meaning under all interpretations and variable assignments. This also extends to conjunctions with more than three conjuncts. Thus, we usually write conjunctions without parentheses. Similarly, disjunctions of the forms $(F \vee G) \vee H$ and $F \vee (G \vee H)$ have the same meaning under all interpretations and variable assignments. This also extends to disjunctions with more than three disjuncts. Thus, we usually write disjunctions without parentheses.

The meaning of a ground term, ground atom, or closed formula does not depend on the variable assignment v. This means that we can speak unambiguously about the meaning of ground terms, ground atoms, and closed formulas with respect to a particular interpretation. Thus, for a ground term t, we usually use t^I instead of $t^{I,v}$; for a ground atom F, we usually use F^I instead of $F^{I,v}$; and for a closed formula G, we usually use G^I instead of $G^{I,v}$. We say that that a closed formula G is true in an interpretation I when G^I is *true*. Likewise, we say that that a closed formula G is false in an interpretation I when G^I is *false*.

Definition 2.13 Let F and G be formulas. We say F and G are *logically equivalent* if $F^{I,v}$ and $G^{I,v}$ are identical for all interpretations I and all variable assignments v. ∎

Definition 2.14 Let F be a closed formula and I an interpretation. We say I is a *model* of F if F^I is *true*; otherwise, we say I is a *countermodel* of F. ∎

Definition 2.15 Let F be a closed formula. We say F is *valid*, denoted by $\models F$, if every interpretation is a model of F; otherwise, we say F is *invalid.* ∎

Definition 2.16 Let F be a closed formula. We say F is *satisfiable* if there exists an interpretation which is a model of F; otherwise, we say F is *unsatisfiable.* ∎

The definitions of model, validity, and satisfiability extend nicely to sets of closed formulas. We define the logical equivalence of sets of closed formulas in terms of models of sets of closed formulas. One interesting result is that a finite set of closed formulas has the same meaning as the conjunction of its elements under all interpretations.

Definition 2.17 Let S be a set of closed formulas and I an interpretation. We say I is a *model* of S if I is a model of every closed formula in S; otherwise, we say I is a *countermodel* of S. ∎

Definition 2.18 Let S be a set of closed formulas. We say S is *valid*, denoted by $\models S$, if every interpretation is a model of S; otherwise, we say S is *invalid.* ∎

Definition 2.19 Let S be a set of closed formulas. We say S is *satisfiable* if there exists an interpretation which is a model of S; otherwise, we say S is *unsatisfiable.* ∎

Definition 2.20 Let S and T be sets of closed formulas. We say S and T are *logically equivalent* if for all interpretations I, I is a model of S iff I is a model of T. ∎

Theorem 2.1 Let $\{ F_1, \ldots, F_n \}$ be a non-empty, finite set of closed formulas and I an interpretation. Then I is a model of $\{ F_1, \ldots, F_n \}$ iff I is a model of $F_1 \wedge \cdots \wedge F_n$.

Proof: Directly from the definitions of model and meaning of $\wedge$. ∎

Another important concept is that of logical consequence.

Definition 2.21 Let S be a set of closed formulas and F a closed formula. We say F is a *logical consequence* of S, denoted by $S \models F$, if every model of S is also a model of F. ∎

Theorem 2.2 Let $\{ F_1, \ldots, F_n \}$ be a non-empty, finite set of closed formulas, and F a closed formula. Then $\{ F_1, \ldots, F_n \} \models F$ iff $\models (F_1 \wedge \cdots \wedge F_n) \rightarrow F$.

Proof: Directly from the definitions of logical consequence and meaning of $\wedge$ and $\rightarrow$. ■

Theorem 2.3* Let S be a set of closed formulas and F a closed formula. Then $S \models F$ iff $S \cup \{ \sim F \}$ is unsatisfiable.

Proof: Directly from the definitions of logical consequence and unsatisfiability. ■

Definition 2.22 Let S and T be sets of closed formulas. We say T is a *logical consequence* of S, denoted by $S \models T$, if every model of S is also a model of T. ■

We conclude our brief discussion on first-order semantics by introducing the notion of a theory. Most texts on logic define theory in the context of provability and proof procedures (or proof systems). However, as the development of efficient proof procedures is the primary goal of automated theorem proving, we find it useful to define a theory independent of proof procedures. This allows us to refer to a theory without regard to a specific proof procedure.

Definition 2.23 A *theory* T is any set of closed formulas. We refer to any closed formula F in T as a *theorem* of T. ■

One important theory is a set of all valid closed formulas.

Definition 2.24 Let T be a theory. We say T is a *predicate calculus* if for every closed formula F, $F \in T$ iff $\models F$. ■

Note that there are many predicate calculi, each differing only in the alphabet used in constructing the closed formulas.

There are two common methods of defining theory.

Definition 2.25 Let S be a set of closed formulas. The *theory* defined by S, denoted by $\mathrm{Th}(S)$, is $\{ F : S \models F \}$. We refer to each closed formula in S an *axiom* of $\mathrm{Th}(S)$. ■

Definition 2.26 Let I be an interpretation. The *theory* defined by I, denoted by $\mathrm{Th}(I)$, is $\{ F : I \text{ is is a model of } F \}$. ■

We now state the theorem proving problem:

> *Given a set S of closed formulas and a closed formula F, is F a logical consequence of S; that is, is F a theorem of* $\mathrm{Th}(S)$*?*

2.3 Proof Procedures

Many systems have been developed to solve the theorem proving problem. These systems can be grouped into a number of broad classes. In fact, two classes, *natural deduction systems* and *axiom systems* (or Hilbert systems), are often used as a formal foundation for first-order logic. For example, Kalish, Montague, and Mar[54] use a natural deduction system to define first-order logic, while Mendelson[71] uses an axiom system to formally describe first-order logic. Another class of systems, *Herbrand systems*, are widely used as a basis for automated theorem provers (proof procedures which are run on computers).

Definition 2.27 A *proof procedure* P is an algorithm, possibly non-terminating, which takes a finite set S of closed formulas and a closed formula F as input and outputs an indication of either "F is logical consequence of S" or "F is not a logical consequence of S". If P generates "F is a logical consequence of S" for a specific S and F, we say F is *derivable* from S by P, denoted by $S \vdash_P F$. If P generates "F is a logical consequence of S" for an empty S and a specific F, we say F is *provable* by P, denoted by $\vdash_P F$, and refer to F as a *theorem* of P. ∎

Proof procedures can be grouped into two general categories: *affirmation procedures* and *refutation procedures*. Consider the theorem proving problem. Affirmation procedures work by trying to show that $S \cup \{ F \}$ is valid, while refutation procedures attempt to show that $S \cup \{ \sim F \}$ is unsatisfiable.

We are often interested in the measuring the efficiency of a particular proof procedure. Since general first-order proof procedures can be at best only partially decidable[1], analyzing the complexity of the full proof procedure is of little benefit, although complexity analysis of specific algorithms used within the proof procedure can be quite useful.

One common way of measuring the efficiency of a proof procedure is to implement the proof procedure in a particular computer programming language and to run the implementation on a number of standard theorems, noting the amount of computer time and memory

[1]The set of valid closed formulas is recursively enumerable but not recursive. This means that there exists no decision procedure for solving the theorem proving problem.

it takes to find the proofs. The TPTP Problem Library of Sutcliffe and Suttner[93,94] provides such a collection of test problems for automated proof procedures. However, there are two problems with this approach: (1) it can be quite time consuming to implement a particular proof procedure and (2) implementation issues, such as the program language and complier used for the implementation, the low-level data structures used in the implementation, and the hardware on which the test problems are run, can complicate the comparison of measurements of different proof procedures.

Another measure commonly used in describing the efficiency of proof procedures, especially in back-of-the-envelope calculations and for arguing the merits of a particular enhancement to a proof procedure, is the size of a proof procedure's search space. Informally, the *search space* of the application of a proof procedure to a finite set S of closed formulas and a closed formula F is the set of all closed formulas "generated" by the proof procedure during the derivation of F from S. The search space of the application of a proof procedure can be divided into a hierarchy of sets S_0, S_1, ... of closed formulas where S_0 is S and S_i is the set of those closed formulas not in any S_j for $0 \leq j < i$ which are derivable in one step from $S_0 \cup \cdots \cup S_{i-1}$. We say a proof procedure P is *breadth-first* if P tends to generate the closed formulas in S_i before generating the closed formulas in S_{i+1}. We say a proof procedure P is *depth-first* if P tends to generate sequences F_i, F_{i+1}, ... of closed formulas where $F_i \in S_i$, $F_{i+1} \in S_{i+1}$, ... Note that our definitions of breadth-first and depth-first are purposely vague; we tend to be fairly loose when referring to a proof procedure as breadth-first or depth-first.

We now state a few important properties of proof procedures.

Definition 2.28 Let P be a proof procedure. We say P is *inconsistent* if there exists a closed formula F such that both $\vdash_P F$ and $\vdash_P \sim F$; otherwise, we say P is *consistent*. ∎

Definition 2.29 Let P be a proof procedure and T a theory. We say P is *sound* if for every closed formula F, $\vdash_P F$ implies $\models F$; otherwise, we say P is *unsound*. We say P is *sound* for T if for every closed formula F, $\vdash_P F$ implies $T \models F$; otherwise, we say P is *unsound* for T. ∎

Definition 2.30 Let P be a proof procedure. We say P is *syntactically complete* if for every closed formula F, either $\vdash_P F$ or $\vdash_P \sim F$; otherwise, we say P is *syntactically incomplete*. ∎

Definition 2.31 Let P be a proof procedure and T a theory. We say P is *logically complete* if for every closed formula F, $\models F$ implies $\vdash_P F$; otherwise, we say P is *logically incomplete.* We say P is *logically complete* for T if for every closed formula F, $T \models F$ implies $\vdash_P F$; otherwise, we say P is *logically incomplete* for T. ∎

Theorem 2.4 Let P be a proof procedure. If P is sound, then P is consistent.

Proof: Directly from the definitions of soundness and consistency. ∎

Theorem 2.5 (Gödel's Completeness Theorem) There exists a sound and logically complete proof procedure.

Proof: First proved by Gödel[41]. Also proved in Ebbinghaus, Flum, and Thomas[32], Gallier[37], and Mendelson[71]. ∎

The use of inference rules is common in describing proof procedures.

Definition 2.32 Let S be a set. An *n-place inference rule* R on S is a mapping $f : X \to S$ where $X \subseteq S^n$. We say $f(s_1, \ldots, s_n)$, if it exists, is a *direct consequence* of s_1, $\ldots$, s_n by application of R. ∎

Inference rules are usually written as follows:

$$\frac{s_1 \quad \cdots \quad s_n}{s}$$

This notation indicates that given elements of the form s_1, $\ldots$, s_n, one can infer the element of form s. For example, the inference rule *modus ponens* on closed formulas can be written as

$$\frac{A \quad A \to B}{B}$$

One can apply modus ponens to the closed formulas p and $p \to q$ to infer the closed formula q. On the other hand, modus ponens does not apply to the closed formulas p and $p \wedge q$ as neither p nor $p \wedge q$ is of the form $A \to B$.

2.4 Normal Forms

Many proof procedures, especially those suited for automation, first transform input formulas into a simpler form before proceeding to show validity or unsatisfiability. When the transformation process generates logically equivalent formulas, we refer to the new form as a *normal form*; otherwise, we refer to the new form as a *standard form*. We now define some common normal and standard forms.

Definition 2.33 A *literal* is an atom other than **T** or $false$ or the negation of an atom other than **T** or $false$. A literal is *positive* if it is an atom; otherwise, it is *negative*. A literal is *ground* if it contains no variables. Two literals L_1 and L_2 are *contradictory* if either L_1 and $\sim L_2$ are identical or $\sim L_1$ and L_2 are identical. ∎

Definition 2.34 Let L be a literal and t a term occurring in L. The *position* p of t in L is the position of t's outer function symbol in the tree of the atom of L. ∎

Definition 2.35 Let F be a formula. We say F is in *conjunctive normal form* if it has the form $F_1 \wedge \cdots \wedge F_n$ where each conjunct F_i has the form $L_1 \vee \cdots \vee L_m$ with each L_i a literal. ∎

Definition 2.36 Let F be a formula. We say F is in *disjunctive normal form* if it has the form $F_1 \vee \cdots \vee F_n$ where each disjunct F_i has the form $L_1 \wedge \cdots \wedge L_m$ with each L_i a literal. ∎

Definition 2.37 Let F be a formula. We say F is in *prenex normal form* if it has the form $Qx_1 \cdots Qx_n\, M$ where each Q_i is either $\forall$ or $\exists$ and M is a quantifier-free formula. We say $Qx_1 \cdots Qx_n$ is the *prefix* of F and M is the *matrix* of F. ∎

Definition 2.38 Let F be a closed formula. We say F is a *clause* if it is **F** or it has the form $\forall x_1 \cdots \forall x_n\, L_1 \vee \cdots \vee L_m$ where each L_i is a literal. We refer to **F** as the *empty clause* and a clause containing one literal as a *unit clause*. A clause is *ground* if it contains no variables. ∎

Theorem 2.6 Let S be a set of clauses. If S contains the empty clause, then S is unsatisfiable.

Proof: Directly from the definitions of empty clause and unsatisfiability. ∎

Definition 2.39 Let $C = \forall x_1 \cdots \forall x_n\, L_1 \vee \cdots \vee L_m$ be a clause, t a term occurring in C, and u a term. The *position* of t in C is $i.p$ where t occurs in L_i and p is the position of t in L_i. We use the notation $C[t]_p$ to indicate that t occurs in C at position p and the notation $C[t \to u]_p$ to denote the clause obtained by replacing the term t at position p by the term u. ∎

Definition 2.40 Let S be a set of clauses. We say S is *variable-disjoint* if for each pair (C_1, C_2) of distinct clauses in S, C_1 and C_2 have no variables in common. ∎

The meaning of a clause is invariant with respect to the ordering and grouping of its literals and the ordering of its universal quantifiers. We, therefore, represent a clause by the set of its literals. For example, the clause $\forall x\, \forall y\, p(x) \vee q(y)$ is represented by the set $\{\, p(x), q(y)\,\}$. We use the empty set to represent the empty clause.

Furthermore, the meaning of a clause is invariant with respect to the naming of its variables. This means that one can convert any set of clauses to an equivalent variable-disjoint set of clauses by simply renaming variables. We, therefore, assume, unless otherwise noted, that a set of clauses is pairwise variable-disjoint.

Definition 2.41 Let F be a closed formula. We say F is in *clause form*[2] if it has the form $C_1 \wedge \cdots \wedge C_n$ where each C_i is a clause. ∎

Definition 2.42 Let C be a clause. We say C is a *Horn clause* if C contains at most one positive literal. ∎

The meaning of a formula in clause form is invariant with respect to the ordering and grouping of its clauses. We, therefore, represent a formula in clause form by the set of its clauses. Combining this with the representation of a clause by the set of its literals, a formula in clause form is represented by the set of sets of its literals. For example, the formula $\forall x\, \forall y\{[p(x) \vee q(x)] \wedge r(y)\}$, which is in clause form, can be represented by $\{\,\{\, p(x), q(x)\,\}, \{\, r(y)\,\}\,\}$.

Theorem 2.7* There exists an algorithm, referred to as *Skolemization*, for converting any closed formula F into a finite set S of closed formulas in clause form such that S is satisfiable iff F is satisfiable.

[2] Clause form is also referred to as *Skolem standard form* or *Skolem conjunctive form* in the literature.

Proof: Proved in Loveland[66]. ■

Since Skolemization replaces the variables of existential quantifiers with Skolem functions (for example the closed formula $\exists x\, p(x)$ is converted into the clause $\{\, p(a)\, \}$), it does not necessarily preserve validity. However, Skolemization does preserve satisfiability. Thus, clause form is better suited for use in refutation proof procedures as they attempt to show the unsatisfiability of a set of closed formulas. Skolemization is discussed further in Chang and Lee[17] and Loveland[66]. Clocksin and Mellish[20] give a Prolog implementation of Skolemization.

2.5 Equality

We consider equality as a binary predicate, normally denoted by the infix operator $=$. We represent the meaning of equality via selected closed formulas, referred to as the *equality axioms*[3].

Definition 2.43 Let L be a first-order language. We say L is a *first-order language with equality* if L contains the 2-place predicate symbol $=$, which we refer to as the *equality symbol*. We refer to a literal whose predicate is the equality symbol as an *equational literal*. We write $=$ as an infix operator and use $s \neq t$ to denote $\sim s = t$. We use *equation* and *inequation* to refer to the unit clauses $\{\, s = t\, \}$ and $\{\, s \neq t\, \}$, respectively. We use $\{\, s \doteq t\, \}$ to denote $\{\, s = t\, \}$ or $\{\, t = s\, \}$. ■

Definition 2.44 The *reflexivity axiom* is the closed formula $\forall x\, x = x$. ■

Definition 2.45 Let f^n be a function symbol. The *replacement axiom* for f^n is the closed formula $\forall x_1 \cdots \forall x_n\, \forall y_1 \cdots \forall y_n (x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$. ■

Definition 2.46 Let p^n be a predicate symbol. The *replacement axiom* for p^n is the closed formula $\forall x_1 \cdots \forall x_n\, \forall y_1 \cdots \forall y_n (x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \rightarrow (p(x_1, \ldots, x_n) \rightarrow p(y_1, \ldots, y_n))$. ■

Definition 2.47 An *equality axiom* is the reflexivity axiom or a replacement axiom. We use $\mathrm{Eq}(S)$ to denote the set of equality axioms for a set S of closed formulas. ■

[3]Note that is also possible to treat equality as a logical symbol in the same manner as $\wedge$, $\vee$, $\sim$, $\rightarrow$, $\equiv$, $\forall$, and $\exists$. For such a treatment, we refer the reader to Boolos and Jeffrey[11].

Note that the replacement axiom is defined for the equality symbol, as well as for other predicate symbols. The replacement axiom for equality is $\forall w \forall x \forall y \forall z (w = y \wedge x = z) \rightarrow (w = x \rightarrow y = z)$.

We are interested in those interpretations in which equality takes on its usual meaning. We refer these interpretation as "normal".

Definition 2.48 Let I be an interpretation. We say I is *normal* if, for all terms t_1 and t_2, $(t_1 = t_2)^I$ is *true* when t_1^I and t_2^I are identical and *false* otherwise. ∎

The definitions of validity, satisfiability, and logical consequence extend to equality.

Definition 2.49 Let F be a closed formula. We say F is *E-valid*, denoted by $\models_= F$, if every normal interpretation is a model of F; otherwise, we say F is *E-invalid.* ∎

Definition 2.50 Let F be a closed formula. We say F is *E-satisfiable* if there exists a normal interpretation which is a model of F; otherwise, we say F is *E-unsatisfiable.* ∎

Definition 2.51 Let S be a set of closed formulas and F a closed formula. We say F is an *E-logical consequence* of S, denoted by $S \models_= F$, if every normal model of S is also a model of F. ∎

Definition 2.52 Let S be a set of closed formulas. We say S is *E-valid*, denoted by $\models_= S$, if every normal interpretation is a model of S; otherwise, we say S is *E-invalid.* ∎

Definition 2.53 Let S be a set of closed formulas. We say S is *E-satisfiable* if there exists a normal interpretation which is a model of S; otherwise, we say S is *E-unsatisfiable.* ∎

Definition 2.54 Let S and T be sets of closed formulas. We say T is an *E-logical consequence* of S, denoted by $S \models_= T$, if every normal model of S is also a model of T. ∎

Theorem 2.8* Let S be a set of closed formulas and F a closed formula. Then $S \models_= F$ iff $S \cup \{ \sim F \}$ is E-unsatisfiable.

Proof: Directly from the definitions of logical consequence and E-unsatisfiability. ∎

Skolemization extends to equality.

Theorem 2.9* Let F be a closed formula and S the Skolemization of F. Then S is E-satisfiable iff F is E-satisfiable.

Proof: Straightforward extension of the proof of Theorem 2.7. ■

The equality axioms capture the properties of equality.

Theorem 2.10 Let S be a set of closed formulas and F a closed formula. Then $S \models_{=} F$ iff $S \cup \mathrm{Eq}(S) \models F$.

Proof: Proved in Fitting[35]. ■

Two other axioms, *symmetry* and *transitivity*, can be used in place of the replacement axioms for equality.

Definition 2.55 The *symmetry axiom* is the closed formula $\forall x \, \forall y \, x = y \rightarrow y = x$. ■

Definition 2.56 The *transitivity axiom*[4] is the closed formula $\forall x \, \forall y \, \forall z (x = y \wedge y = z) \rightarrow x = z$. ■

Theorem 2.11 The sets of closed formulas $\{ \forall x \, x = x, \forall x \, \forall y \, x = y \rightarrow y = x, \forall x \, \forall y \, \forall z \, (x = y \wedge y = z) \rightarrow x = z \}$ and $\{ \forall x \, x = x, \forall w \, \forall x \, \forall y \, \forall z (w = y \wedge x = z) \rightarrow (w = x \rightarrow y = z) \}$ are logically equivalent.

Proof: Directly from the definition of logical equivalence. ■

Equality can be used to define theories.

Definition 2.57 Let E be a set of equations and s and t terms. We say s and t are *E-equivalent*, denoted by $s =_E t$, if $E \models_{=} \{ s = t \}$. ■

Definition 2.58 Let E be a set of equations. The *theory* defined by E, denoted by $\mathrm{Th}(E)$, is $\{ s = t : s =_E t \}$. We refer to $\mathrm{Th}(E)$ as an *equational theory.* ■

[4]The transitivity axiom is sometimes defined as the closed formula $\forall x \, \forall y \, \forall z (x = y \wedge x = z) \rightarrow y = z$.

2.6 Unification

Another technique useful in automated theorem proving is the process of making a set of terms or clauses identical by appropriate substitution of variables by terms. This process is referred to as unification. For example, the set of terms $\{ f(w, a), f(g(b), x), f(y, z) \}$ can be can be made identical by substituting $g(b)$ for w and y and a for x and z.

Definition 2.59 Let T be a set of terms and X the set of variables in T. A *substitution* θ, denoted by $\{ x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \}$, is a finite subset of $X \times T$ such that $x_1, \ldots, x_n$ are distinct and each x_i is distinct from t_i. We refer to $x_i \mapsto t_i$ as the *binding* of x_i in θ, x_i as the *binding name* of $x_i \mapsto t_i$, and t_i as the *binding term* of $x_i \mapsto t_i$. The *identity substitution*, denoted by ε, is the empty set. A substitution is *ground* if all of its binding terms are ground. A substitution is *variable-pure* if all of its binding terms are variables. ■

We now define the application of a substitution to a term, literal, or clause.

Definition 2.60 Let E be a term or literal and $\theta = \{ x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \}$ a substitution. The *instance* of E by θ, denoted by $E\theta$, is the term or literal obtained by simultaneously replacing each variable x_i in E by the term t_i. We say $E\theta$ is a *proper instance* of E if the application of θ to E replaces some variable x_i in E with a non-variable term. ■

Definition 2.61 Let $C = \forall x_1 \cdots \forall x_n\, F$ be a clause and $\theta = \{ x_1 \mapsto t_1, \ldots, x_m \mapsto t_m \}$ a substitution. The *instance* of C by θ, denoted by $C\theta$, is the clause $\forall y_1 \cdots \forall y_k\, F\theta$ where $F\theta$ is obtained by simultaneously replacing each variable x_i in F by the term t_i and $y_1, \ldots, y_k$ are the variables occurring in $F\theta$. We say $C\theta$ is a *proper instance* of C if the application of θ to C replaces some variable x_i in C with a non-variable term. ■

Definition 2.62 Let S be a set of terms, literals, or clauses and θ a substitution. The *instance* of S by θ, denoted by $S\theta$, is the set $\{ E\theta : E \in S \}$. We say $S\theta$ is a *proper instance* of S if there exists an E in S such that $E\theta$ is a proper instance of E. ■

A basic operation on substitutions is composition.

Definition 2.63 Let $\theta = \{ x_1 \mapsto s_1, \ldots, x_n \mapsto s_n \}$ and $\sigma = \{ y_1 \mapsto t_1, \ldots, y_m \mapsto t_m \}$ be substitutions. The *composition* of θ and σ, denoted by $\theta\sigma$, is $\{ x_1 \mapsto t_1\sigma, \ldots, x_n \mapsto t_n\sigma,$

$y_1 \mapsto t_1, \ldots, y_m \mapsto t_m \} - \{ x_i \mapsto t_i\sigma \ : \ x_i$ and $t_i\sigma$ are identical $\} - \{ y_j \mapsto t_j \ : \ y_j \in \{ x_1, \ldots, x_n \} \}$. ∎

For example, let $\theta = \{ x \mapsto f(y), z \mapsto w \}$ and $\sigma = \{ w \mapsto z, x \mapsto g(a), y \mapsto b \}$. $\theta\sigma = \{ x \mapsto f(b), z \mapsto z, x \mapsto g(a), y \mapsto b, w \mapsto z \} - \{ z \mapsto z \} - \{ x \mapsto g(a) \} = \{ x \mapsto f(b), y \mapsto b, w \mapsto z \}$.

We now present a number of basic properties of substitutions.

Theorem 2.12 Let E be a term, literal, or clause, θ, σ, and η substitutions, and ε the identity substitution. Then,

(a) $\theta\varepsilon = \varepsilon\theta = \theta$.
(b) $(E\theta)\sigma = E(\theta\sigma)$.
(c) $\theta = \sigma$ iff $t\theta = t\sigma$ for all terms t.
(d) $(\theta\sigma)\eta = \theta(\sigma\eta)$.

Proof: Proved in Loveland[66]. ∎

Definition 2.64 Let E and F be terms, literals, or clauses. We say E and F are *variants*, denoted by $E \approx F$, if there exist substitutions θ and σ such that E is $F\theta$ and F is $E\sigma$. ∎

Definition 2.65 Let E be a term, literal, or clause and V the set of variables occurring in E. A variable-pure substitution $\{ x_1 \mapsto y_1, \ldots, x_n \mapsto y_n \}$ is a *renaming substitution* for E if $\{ x_1, \ldots, x_n \} \in V$, y_1, $\ldots$, y_n are distinct, and $(V - \{ x_1, \ldots, x_n \}) \cap \{ y_1, \ldots, y_n \}$ is empty. ∎

Theorem 2.13 Let E and F be terms, literals, or clauses. If $E \approx F$, then there exists substitutions θ and σ such that E is $F\theta$ and F is $E\sigma$ where θ is a renaming substitution for F, and σ is a renaming substitution for E.

Proof: Directly from the definitions of variant and renaming substitution. ∎

We now define unification.

Definition 2.66 Let $S = \{ E_1, \ldots, E_n \}$ be a finite set of terms or literals. A substitution θ is a *unifier* of S if $E_1\theta$, $\ldots$, $E_n\theta$ are identical. We say S is *unifiable* is there exists a unifier of S. ∎

Definition 2.67 Let S be a finite set of terms or literals and θ a unifier of S. We say θ is a *most general unifier* of S if for each unifier σ of S there exists a substitution η such that $E\sigma = E\theta\eta$ for all $E \in S$. ∎

Theorem 2.14 (Unification Theorem) Let S be a finite set of terms or literals. There exists an algorithm which returns a most general unifier of S if S is unifiable or reports that S is not unifiable otherwise.

Proof: Proved in Fitting[35], Gallier[37], and Loveland[66]. ∎

A unifiable set of terms or literals has a unique most general unifier up to the naming of variables.

Theorem 2.15 Let S be a unifiable set of terms or literals and θ and σ most general unifiers of S. Then $E\theta \approx E\sigma$ for all E in S.

Proof: Proved in Fitting[35], Gallier[37], and Loveland[66]. ∎

Definition 2.68 Let L_1 and L_2 be literals. We say L_1 and L_2 *link* if either L_1 and $\sim L_2$ or $\sim L_1$ and L_2 unify. ∎

Definition 2.69 Let L_1 and L_2 be literals. We say L_2 is a *contradictory instance* of L_1 if either L_2 is an instance of $\sim L_1$ or $\sim L_2$ is an instance of L_1.

Definition 2.70 Let C_1 and C_2 be clauses. We say C_1 *subsumes* C_2 if some subset of C_2 is an instance of C_1.

Unification is discussed further in Chang and Lee[17], Fitting[35], Lloyd[64], and Loveland[66].

2.7 The Skolem-Herbrand-Gödel Theorem

Refutation proof procedures work by showing that there exists no interpretation which satisfies a set of closed formulas. This, in general, is difficult to show since the domain of an arbitrary interpretation can be any arbitrary set. However, if each formula in the set of closed formulas is a clause, the proof procedure need only examine ground instances of the clauses, as shown by Herbrand[45]. We refer to the set of ground terms as the *Herbrand universe* and interpretations based on the Herbrand universe as *Herbrand interpretations.* A consequence of Herbrand's work, known as the Skolem-Herbrand-Gödel theorem[5], is that a set of clauses is unsatisfiable if, and only if, it has an unsatisfiable finite subset of ground instances. Thus, a refutation proof procedure can first convert a set of closed formulas into a set of clauses and then attempt to find an unsatisfiable finite subset of ground instances. Refutation proof procedures which follow this process are referred to as *instance-based* refutation proof procedures.

Definition 2.71 Let A be an alphabet. The *Herbrand universe* of A is the set of all ground terms formed from the function symbols of A. If there is no constant in A, the alphabet A is extended by adding a constant symbol f_i^0 to A. In this case, we refer to f_i^0 as the *Herbrand constant* of A. ∎

Definition 2.72 Let $I = \langle D, I_f, I_p \rangle$ be an interpretation. We say I is a *Herbrand interpretation* if

(a) D is the Herbrand universe.
(b) I_f maps each function symbol f_n into the function $(t_1, \ldots, t_n) \rightarrow f^n(t_1, \ldots, t_n)$. ∎

Theorem 2.16 A set of clauses is unsatisfiable iff it has no Herbrand model.

Proof: Proved in Gallier[37]. Also, stated in Fitting[35] with proof left as an exercise. ∎

Theorem 2.17* **(Skolem-Herbrand-Gödel Theorem)** A set S of clauses is unsatisfiable iff there exists an unsatisfiable finite subset of ground instances of S.

Proof: Proved in Gallier[37] and Loveland[66]. ∎

The Skolem-Herbrand-Gödel theorem extends to equality.

[5]The Skolem-Herbrand-Gödel theorem is sometimes incorrectly referred to as Herbrand's theorem in the literature. One should note that Herbrand's theorem deals with provability, a proof-theoretic concept, while the Skolem-Herbrand-Gödel theorem deals with unsatisfiability, a semantic concept. For further discussion on the difference between Herbrand's theorem and the Skolem-Herbrand-Gödel theorem, we refer the reader to Gallier[37] and Herbrand[46].

Theorem 2.18* **(Skolem-Herbrand-Gödel Theorem for Equality)** A set S of clauses is E-unsatisfiable iff there exists an E-unsatisfiable finite subset of ground instances of S.

Proof: Proved in Andrews[2] and Gallier[37]. ∎

2.8 Methods for Handling Equality

Based on Theorem 2.10, the obvious solution to the theorem proving problem for formulas with equality is to simply use the equality axioms. While theoretically correct, this does not work in practice for most proof procedures, as even simple equality theorems require substantial time to prove. Many techniques have been developed to address the performance problems in proving equality theorems. Some of these are paramodulation, Brand's E-modification, term rewriting, ordered rewriting, completion, and specialized unification.

2.8.1 Paramodulation

For most proof procedures, the inefficiency experienced when using the equality axioms is primarily due to the large number of consequences resulting from the replacement axioms. One technique for addressing the large number of consequences is to replace the replacement axioms with the following inference rule (recall that $C[t \rightarrow r]_p$ denotes the clause obtained by replacing the term t at position p by the term r):

Paramodulation: $$\frac{C[t]_p \qquad \{\, l \doteq r \,\} \cup D}{(C[t \rightarrow r]_p \cup D)\theta}$$ where θ is a most general unifier of t and l

For example, consider the following two variable-disjoint clauses: $C_1 = \{\, p(f(x,y)), g(x) = f(x,b) \,\}$ and $C_2 = \{\, q(f(g(a),b)) \,\}$. Taking C as $\{\, q(f(g(a),b)) \,\}$, t as $g(a)$, p as 1.1.1, l as $g(x)$, r as $f(x,b)$, and D as $\{\, p(f(x,y)) \,\}$, one obtains the the clause $\{\, p(f(a,y)), q(f(f(a,b),b)) \,\}$. Taking C as $\{\, q(f(g(a),b)) \,\}$, t as $f(g(a),b)$, p as 1.1, l as $f(x,b)$, r as $g(x)$, and D as $\{\, p(f(x,y)) \,\}$, one obtains the clause $\{\, p(f(g(a),y)), q(g(g(a))) \,\}$.

Paramodulation has been studied much in conjunction with resolution proof procedures, and many refinements to paramodulation have been developed. When combining paramodulation with resolution proof procedures, one usually includes the *functionally reflexive axiom*, $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$, for each function symbol f (except for individual constants). Brand[14] showed that the functionally reflexive axioms are unnecessary when paramodulation is used with unrestricted resolution. However, the functionally reflexive axioms are required for many of the restrictions placed on resolution.

While paramodulation works well with resolution proof procedures, it is not well suited for hyper-linking. The reason is that paramodulation combines literals from two clauses leading to an exponential explosion in the number of clauses generated. This is not in line with the instance-based nature of hyper-linking, which does not combine literals from differing clauses.

Further discussion on paramodulation can be found in Loveland[66] and Chang and Lee[17].

2.8.2 Brand's E-Modification

Brand[14] developed an alternative to paramodulation based on "pulling out" non-variable terms from function and predicate arguments. For example, consider the set of clauses $\{\{p(a)\},\{\sim p(b)\},\{a=b\}\}$ which is E-unsatisfiable. Pulling out the argument a from $p(a)$ gives the set of clauses $\{\{a \neq x, p(x)\},\{\sim p(b)\},\{a=b\}\}$ which is unsatisfiable under all interpretations. Brand generalized the method of pulling out terms into a number of different transformations. One such transformation is the E-modification.

Definition 2.73 Let C be a clause. If all non-variable subterms of C occur as arguments of $=$ or $\neq$, the *E-modification* of C is C. Otherwise, let s be a non-variable subterm of C at position p which is not the argument of $=$ or $\neq$. The *E-modification* of C, denoted by E-mod(C), is the E-modification of $\{s \neq x\} \cup C[s \rightarrow x]_p$ (recall that $C[s \rightarrow x]_p$ denotes the clause obtained by replacing the term s at position p by the variable x) where x is a variable not occurring in C. ∎

For example, the E-modification of $\{f(a,b)=f(c,d)\}$ is $\{d \neq w, c \neq x, b \neq y, a \neq z, f(z,y)=f(x,w)\}$, and the E-modification of $\{f(f(x,y),z)=f(x,f(y,z))\}$ is $\{f(y,z) \neq v, f(x,y) \neq w, f(w,z)=f(x,v)\}$.

We note that the E-modification of a clause is unique up to the renaming of variables.

Theorem 2.19 Let C and D be clauses. Then E-mod$(C) \approx$ E-mod(D) iff $C \approx D$.

Proof: Directly from the definitions of E-modification and variant. ∎

We define the E-modification of a set of clauses to be the E-modifications of each clause in the set.

Definition 2.74 Let S be a set of clauses. The *E-modification* of S, denoted by E-mod(S), is $\{\, \text{E-mod}(C) : C \in S \,\}$. ∎

E-modification takes a set of clauses which is E-unsatisfiable and generates a set of clauses which, when the reflexivity and transitivity axioms are added, is unsatisfiable.

Theorem 2.20* Let S be a set of clauses and I an interpretation. Then I is a normal model of S iff I is a model of E-mod$(S) \cup \{\, \{\, x = x \,\}, \{\, x \neq y, x \neq z, y = z \,\} \,\}$.

Proof: See Brand[14]. ∎

Corollary 2.21 Let S be a set of clauses. Then S is E-unsatisfiable iff E-mod$(S) \cup \{\, \{\, x = x \,\}, \{\, x \neq y, x \neq z, y = z \,\} \,\}$ is unsatisfiable.

Proof: Directly from Theorem 2.20. ∎

Corollary 2.22 Let S be a set of clauses. Then every model of E-mod$(S) \cup \{\, \{\, x = x \,\}, \{\, x \neq y, x \neq z, y = z \,\} \,\}$ is a normal model.

Proof: Directly from Theorem 2.20. ∎

Brand's E-modification offers an attractive alternative to the equality axioms for converting an E-unsatisfiable set of clauses into an unsatisfiable set of clauses.

2.8.3 Term Rewriting

Another method for reducing the large number of consequences generated when proving equality theorems is to simplify closed formulas. This process is referred to as *algebraic simplification*. For example, consider the equations

$$
\begin{array}{lrcl}
(1) & x + 0 & = & x \\
(2) & x + s(y) & = & s(x + y) \\
(3) & x * 0 & = & 0 \\
(4) & x * s(y) & = & (x * y) + x
\end{array}
$$

Using these equations, one can simplify $s(0) * s(s(0))$ to $s(s(0))$ as follows:

$$\begin{array}{llcl}
\text{by equation (4)} & s(0) * s(s(0)) & = & (s(0) * s(0)) + s(0) \\
\text{by equation (4)} & & = & ((s(0) * 0) + s(0)) + s(0) \\
\text{by equation (3)} & & = & (0 + s(0)) + s(0) \\
\text{by equation (2)} & & = & s(0 + 0) + s(0) \\
\text{by equation (1)} & & = & s(0) + s(0) \\
\text{by equation (2)} & & = & s(s(0) + 0) \\
\text{by equation (1)} & & = & s(s(0))
\end{array}$$

The formal study of algebraic simplification is referred to as *term rewriting*.

Definition 2.75 Let $\rightarrow$ be a binary relation on terms and s and t terms. We use $s \rightarrow^* t$ to indicate that there exists a sequence, possibly of length zero, $s \rightarrow \cdots \rightarrow t$, $s \leftrightarrow t$ to denote $s \rightarrow t$ or $t \rightarrow s$, and $s \leftrightarrow^* t$ to indicate that there exists a sequence, possibly of length zero, $s \leftrightarrow \cdots \leftrightarrow t$. We use $s \not\rightarrow t$ to denote not $s \rightarrow t$, $s \not\rightarrow^* t$ to indicate that there exists no sequence $s \rightarrow \cdots \rightarrow t$, $s \not\leftrightarrow t$ to denote not $s \rightarrow t$ and not $t \rightarrow s$, and $s \not\leftrightarrow^* t$ to indicate that there exists no sequence $s \leftrightarrow \cdots \leftrightarrow t$. ∎

Definition 2.76 Let T be a set of terms. A *term rewriting system* R is a subset of $T \times T$. We refer to each ordered pair (s, t) in R, denoted by $s \rightarrow t$, as a *rewrite rule* of R. ∎

Definition 2.77 Let R be a term rewriting system and s and t terms. We say s *rewrites* to t under R, denoted by $s \rightarrow_R t$, if there exists a rewrite rule $l \rightarrow r$ in R, a substitution θ, and a position p such that $s[l\theta]_p$ and t is $s[l\theta \rightarrow r\theta]_p$ (recall that $s[l\theta \rightarrow r\theta]_p$ denotes the term obtained by replacing the term $l\theta$ at position p by the term $r\theta$). We refer to $\rightarrow_R$ as the *rewrite relation* of R and $l\theta$ as the *redex* of $s \rightarrow_R t$. ∎

Definition 2.78 Let R be a term rewriting system and s and t terms. We say s is *R-reducible* if there exists a term u such that $s \rightarrow_R u$; otherwise, we say s is *R-irreducible*. We say t is an *R-normal form* of s is $s \rightarrow_R^* t$ and t is R-irreducible. ∎

Definition 2.79 Let R be a term rewriting system. We say R is *reduced* if for all rewriting rules $l \rightarrow r$ in R, l is R'-irreducible and r is R-irreducible where R' is $R - \{ l \rightarrow r \}$. ∎

When rewriting a given term, there often are many subterms which can be rewritten, thus allowing a choice of which subterm to rewrite. We refer to an algorithm for rewriting terms as a *term rewriting strategy.* Two common term rewriting strategies are *innermost rewriting* and *outermost rewriting.*

Definition 2.80 A *term rewriting strategy* is an algorithm, possibly non-terminating, which takes a term rewriting system R and a term t_0 as input and outputs a sequence $t_0 \rightarrow_R t_1 \rightarrow_R \cdots$ of terms. ■

Definition 2.81 Let S be a term rewriting strategy. We say S is an *innermost term rewriting strategy* if for all term rewriting systems R and terms t_0, running S on (R, t_0) generates a sequence $t_0 \rightarrow_R t_1 \rightarrow_R \cdots$ of terms with $t_{i-1} = t[u]_p$ and $t_i = t[v]_p$ for $i \geq 1$ such that all proper subterms of u are R-irreducible. ■

Definition 2.82 Let S be a term rewriting strategy. We say S is an *outermost term rewriting strategy* if for all term rewriting systems R and terms t_0, running S on (R, t_0) generates the sequence $t_0 \rightarrow_R t_1 \rightarrow_R \cdots$ of terms with $t_{i-1} = t[u]_p$ and $t_i = t[v]_p$ for $i \geq 1$ such that all subterms of t_{i-1} for which u at position p is a proper subterm are R-irreducible. ■

Term rewriting extends to clauses and sets of clauses in a straightforward manner.

Definition 2.83 Let R be a term rewriting system and C and D clauses. We say C *rewrites* to D under R, denoted by $C \rightarrow_R D$, if there exist terms s and t and position p such that $s \rightarrow_R t$, $C[s]_p$, and D is $C[s \rightarrow t]_p$ (recall that $C[s \rightarrow t]_p$ denotes the clause obtained by replacing the term s at position p by the term t). ■

Definition 2.84 Let R be a term rewriting system and C and D clauses. We say C is *R-reducible* if there exists a clause C' such that $C \rightarrow_R C'$; otherwise, we say C is *R-irreducible.* We say D is an *R-normal form* of C if $C \rightarrow_R^* D$ and D is R-irreducible. ■

Definition 2.85 Let R be a term rewriting system and S and T sets of clauses. We say S *rewrites* to T under R, denoted by $S \rightarrow_R T$, if there exist clauses C and D such that $C \rightarrow_R D$, S contains C, and T is $S - \{ C \} \cup \{ D \}$. ■

Definition 2.86 Let R be a term rewriting system and S and T sets of clauses. We say S is *R-reducible* if there exists a set S' of clauses such that $S \rightarrow_R S'$; otherwise, we say S is *R-irreducible.* We say T is an *R-normal form* of S if $S \rightarrow_R^* T$ and T is R-irreducible. ■

The concept of term rewriting can be applied to set of equations. We refer to this as *equational rewriting*. Equational rewriting provides a method for deriving logical consequences of sets of equations as shown by Birkhoff[9].

Definition 2.87 Let E be a set of equations and s and t terms. We say s *rewrites* to t under E, denoted by $s \rightarrow_E t$, if there exists an equation $\{\, l \doteq r \,\}$ in R, a substitution θ, and a position p such that $s[l\theta]_p$ and t is $s[l\theta \rightarrow r\theta]_p$ (recall that $s[l\theta \rightarrow r\theta]_p$ denotes the term obtained by replacing the term $l\theta$ at position p by the term $r\theta$). We refer to $\rightarrow_E$ as the *rewrite relation* of E and $l\theta$ as the *redex* of $s \rightarrow_E t$. ∎

Definition 2.88 Let E be a set of equations and C and D clauses. We say C *rewrites* to D under E, denoted by $C \rightarrow_E D$, if there exist terms s and t and position p such that $s \rightarrow_E t$, $C[s]_p$, and D is $C[s \rightarrow t]_p$ (recall that $C[s \rightarrow t]_p$ denotes the clause obtained by replacing the term s at position p by the term t). ∎

Definition 2.89 Let E be a set of equations and C and D clauses. We say C is E-*reducible* if there exists a clause C' such that $C \rightarrow_E C'$; otherwise, we say C is E-*irreducible*. We say D is an E-*normal form* of C if $C \rightarrow_E^* D$ and D is E-irreducible. ∎

Definition 2.90 Let E be a set of equations and S and T sets of clauses. We say S *rewrites* to T under E, denoted by $S \rightarrow_E T$, if there exist clauses C and D such that $C \rightarrow_E D$, S contains C, and T is $S - \{\, C \,\} \cup \{\, D \,\}$. ∎

Definition 2.91 Let E be a set of equations and S and T sets of clauses. We say S is E-*reducible* if there exists a set S' of clauses such that $S \rightarrow_E S'$; otherwise, we say S is E-*irreducible*. We say T is an E-*normal form* of S if $S \rightarrow_E^* T$ and T is E-irreducible. ∎

Theorem 2.23 (Birkhoff's Theorem) Let E be a set of equations. Then for all terms s and t, $s =_E t$ iff $s \leftrightarrow_E^* t$.

Proof: Proved in Birkhoff[9]. ∎

Term rewriting also provides a method for deriving logical consequences of sets of equations.

Definition 2.92 Let R be a term rewriting system. The set of equations of R, denoted by $R^=$ is $\{\, \{\, s = t \,\} : s \rightarrow t \in R \,\}$. We use $s =_R t$ to denote $R^= \models_= \{\, s = t \,\}$. ∎

Definition 2.93 Let R be a term rewriting system. The *theory* defined by R, denoted by $\mathrm{Th}(R)$, is $\mathrm{Th}(R^=)$. ∎

Theorem 2.24 Let R be a term rewriting system. Then for all terms s and t, $s =_R t$ iff $s \leftrightarrow_R^* t$.

Proof: Directly from Birkhoff's Theorem (Theorem 2.23). ∎

Theorem 2.25 Let R be a term rewriting system and C and D clauses such that $C \to_R^* D$. Then $R^= \cup \{C\} \models_= D$.

Proof: Directly from the definition of rewriting and E-logical consequence. ∎

Definition 2.94 Let E be a set of equations and R a term rewriting system. We say E and R are *equivalent* if $\mathrm{Th}(E) = \mathrm{Th}(R)$. ∎

We now define some properties of term rewriting systems. We present the properties as properties of binary relations on terms rather than as properties of term rewriting systems. This allows the definitions to apply to rewrite relations of systems other than term rewriting systems. It is also useful to associate these properties with a term rewriting system. Thus, we sometimes say a term rewriting system has a particular property when we actually mean that its underlying rewrite relation has the property. For example, we may say a term rewriting system R is confluent when we actually mean that the rewrite relation $\to_R$ is confluent.

Definition 2.95 Let R be a binary relation on terms and s and t terms. We write $s \uparrow_R t$ to indicate that there exists a term u such that $u\ R^*\ s$ and $u\ R^*\ t$. We write $s \downarrow_R t$ to indicate that there exists a term u such that $s\ R^*\ u$ and $t\ R^*\ u$. ∎

Definition 2.96 Let R be a binary relation on terms. We say R is *confluent* if for all terms s and t, $s \uparrow_R t$ implies $s \downarrow_R t$. We say R is *ground confluent* if for all ground terms u and terms s and t, $u\ R^*\ s$ and $u\ R^*\ t$ implies $s \downarrow_R t$. We say R is *locally confluent* if for all terms u and terms s and t, $u\ R\ s$ and $u\ R\ t$ implies $s \downarrow_R t$. ∎

Definition 2.97 Let $\to$ be a binary relation on terms. We say $\to$ has the *Church-Rosser property* if for all terms s and t, $s \leftrightarrow^* t$ iff $s \downarrow_\to t$. ∎

Theorem 2.26 Let R be a binary relation on terms. Then R has the Church-Rosser property iff R is confluent.

Proof: Proved in Plaisted[82]. ∎

Definition 2.98 Let R be a binary relation on terms. We say R is *terminating* if there is no infinite sequence $s_1 \; R \; s_2 \; R \cdots$; otherwise, we say R is *non-terminating*. ∎

Theorem 2.27 (Newman's Lemma) Let R be a binary relation on terms. Then R is confluent iff R is locally confluent and terminating.

Proof: Proved in Plaisted[82]. ∎

Definition 2.99 Let R be a term rewriting system. We say R is *canonical* if R is reduced, terminating, and confluent. We say R is *ground canonical* if R is reduced, terminating, and ground confluent. ∎

Dershowitz and Jouannaud[29] and Plaisted[82] present nice surveys on term rewriting.

2.8.4 Termination of Term Rewriting Systems

When using term rewriting within a proof procedure, it is useful to know that the term rewriting system is terminating, as this ensures that the proof procedure never loops rewriting a single term. Note that there are ways of working with non-terminating term rewriting systems in a proof procedure, such as placing a bound on the number of rewrite steps allowed when rewriting a term; however, these methods are usually quite cumbersome and often complicate showing completeness of the proof procedure. Termination orderings offer an attractive way of showing that a term rewriting system is terminating.

Definition 2.100 Let $>$ be a binary relation on a set S. We say $>$ is a *partial ordering* on S if $>$ is irreflexive and transitive. We use $s \not> t$ to denote not $s > t$. ∎

Definition 2.101 Let $>$ be a partial ordering on a set S. We say $>$ is *total* if for all distinct elements x and y in S, either $x > y$ or $y > x$. ∎

Definition 2.102 Let $\{ s = t \}$ be an equation and $>$ a partial ordering on terms. We say $\{ s = t \}$ is *orderable under* $>$ if $s > t$ or $t > s$; otherwise, we say $\{ s = t \}$ is *unorderable under* $>$. ∎

Definition 2.103 Let $>$ be a partial ordering on a set S. We say $>$ is *well-founded* if there exists no infinite sequence $s_1 > s_2 > \cdots$ of elements of S. ■

Definition 2.104 Let E be a set of equations and $>$ be a partial ordering on terms. We say $>$ is *complete* for E if $>$ is total on all E-equivalent ground terms. We say $>$ is *complete* if $>$ is total on ground terms. ■

Definition 2.105 Let $>$ be a partial ordering on terms and R a term rewriting system. We say $>$ *orders* R if for all $l \to r$ in R, $l > r$. ■

Definition 2.106 Let $>$ be a partial ordering on terms. We say $>$ is a *termination ordering*[6] if every term rewriting system ordered by $>$ is terminating. ■

Definition 2.107 Let $>$ be a partial ordering on terms. We say $>$ has the *full invariance property* if for all terms s and t and substitutions θ, $s > t$ implies $s\theta > t\theta$. ■

Definition 2.108 Let $>$ be a partial ordering on terms. We say $>$ has the *replacement property* if for all terms s, t, and u and positions p, $s > t$ and $u[s]_p$ implies $u[s]_p > u[s \to t]_p$ (recall that $u[s \to t]_p$ denotes the term obtained by replacing the term s at position p by the term t). ■

Definition 2.109 Let $>$ be a partial ordering. We say $>$ is a *reduction ordering* if $>$ is well-founded, has the full invariance property, and has the replacement property. ■

Theorem 2.28 All reduction orderings are termination orderings.

Proof: Proved in Plaisted[82]. ■

Another class of orderings useful in showing termination of term rewriting systems are the simplification orderings.

Definition 2.110 Let $>$ be a partial ordering on terms. We say $>$ has the *subterm property*[7] if for all terms s and proper subterms t of s, $s > t$. ■

---

[6]Our definition of termination ordering differs from that of Plaisted[82]. We use reduction ordering to refer to Plaisted's termination ordering.

[7]In the literature the subterm property, simplification ordering, and quasi-simplification ordering are sometimes defined on terms containing function symbols with varying arity, as well as on terms containing function symbols with fixed arity. For simplicity, we restrict our definitions to terms containing only fixed arity function symbols.

Definition 2.111 Let $>$ be a partial ordering on terms. We say $>$ is a *simplification ordering* if $>$ has the replacement and subterm properties. ■

Theorem 2.29 All simplification orderings are well-founded.

Proof: Proved in Plaisted[82]. ■

It is sometimes easier to show termination of a term rewriting system using quasi-orderings rather than partial orderings.

Definition 2.112 Let $\geq$ be a binary relation on a set S. We say $\geq$ is a *quasi-ordering* if $\geq$ is reflexive and transitive. We use $s \not\geq t$ to denote not $s \geq t$. ■

Definition 2.113 Let $\geq$ be a quasi-ordering on a set S. We say the partial ordering $>$ on S is the *strict part* of $\geq$ if for all $x, y \in S$, $x > y$ iff $x \geq y$ and $y \not\geq x$. We say the binary relation $=$ on S is the *equivalence part* of $\geq$ if for all $x, y \in S$, $x = y$ iff $x \geq y$ and $y \geq x$. ■

Definition 2.114 Let $\geq$ be a quasi-ordering on a set S. We say $\geq$ is *total* if for all elements x and y in S, either $x \geq y$ or $y \geq x$. ■

Definition 2.115 Let $\{ s = t \}$ be an equation and $\geq$ a quasi-ordering on terms. We say $\{ s = t \}$ is *orderable under* $\geq$ if $s \geq t$ or $t \geq s$; otherwise, we say $\{ s = t \}$ is *unorderable under* $\geq$. ■

Definition 2.116 Let $\geq$ be a quasi-ordering on terms. We say $\geq$ has the *replacement property* if for all terms s, t, and u and positions p, $s \geq t$ and $u[s]_p$ implies $u[s]_p \geq u[s \to t]_p$ (recall that $u[s \to t]_p$ denotes the term obtained by replacing the term s at position p by the term t). ■

Definition 2.117 Let $\geq$ be a quasi-ordering on terms. We say $\geq$ has the *subterm property* if for all terms s and subterms t of s, $s \geq t$. ■

Definition 2.118 Let $\geq$ be a quasi-ordering on terms. We say $\geq$ is a *quasi-simplification ordering* if $\geq$ has the replacement and subterm properties. ■

Definition 2.119 Let $\geq$ be a quasi-ordering on terms. We say $\geq$ has the *full invariance property* if for all terms s and t and substitutions θ, $s \geq t$ implies $s\theta \geq t\theta$. ■

Theorem 2.30 Let T be any set of terms with a finite number of function symbols. Then all quasi-simplification orderings on T have well-founded strict parts.

Proof: Proved in Dershowitz[25]. ∎

Theorem 2.31 Let $\geq$ be a quasi-simplification ordering with strict part $>$. If $>$ has the full invariance property, then $>$ is a termination ordering.

Proof: Proved in Plaisted[82]. ∎

There are many different partial and quasi-orderings used to show termination of term rewriting systems. Some examples are the Knuth-Bendix orderings[58], the polynomial orderings of Lankford[60,61], and the recursive path orderings (see Dershowitz[25] or Plaisted[82]).

As an example, we present the lexicographic path ordering of Kamin and Lévy[55] as a quasi-ordering[8]. The lexicographic path ordering is defined in terms of a *quasi-precedence ordering* on function symbols and the *lexicographic extension* of a quasi-ordering.

Definition 2.120 Let F be a set of function symbols. We say a quasi-ordering on F is a *quasi-precedence ordering.* ∎

Definition 2.121 Let $\geq$ be a quasi-ordering on terms with strict part $>$ and equivalence part $=$. The quasi-ordering $\geq_{\text{lex}}$ on tuples of terms with strict part $>_{\text{lex}}$ and equivalence part $=_{\text{lex}}$ defined by the following inference rules is the *lexicographic extension* of $\geq$:

$$(1) \quad \frac{\mathit{true}}{() =_{\text{lex}} ()}$$

$$(2) \quad \frac{\mathit{true}}{(s_1, \ldots, s_m) >_{\text{lex}} ()}$$

$$(3) \quad \frac{s_1 > t_1}{(s_1, \ldots, s_m) >_{\text{lex}} (t_1, \ldots, t_n)}$$

$$(4) \quad \frac{s_1 = t_1 \quad (s_2, \ldots, s_m) =_{\text{lex}} (t_2, \ldots, t_n)}{(s_1, \ldots, s_m) =_{\text{lex}} (t_1, \ldots, t_n)}$$

$$(5) \quad \frac{s_1 = t_1 \quad (s_2, \ldots, s_m) >_{\text{lex}} (t_2, \ldots, t_n)}{(s_1, \ldots, s_m) >_{\text{lex}} (t_1, \ldots, t_n)}$$

∎

[8] The lexicographic path ordering can also be defined as a partial ordering. For such a definition, we refer the reader to Plaisted[82].

Definition 2.122 Let $\geq$ be a quasi-precedence ordering with strict part $>$ and equivalence part $=$. The quasi-ordering $\geq_{\text{lpo}}$ with strict part $>_{\text{lpo}}$ and equivalence part $=_{\text{lpo}}$ defined by the following inference rules is the *lexicographic path ordering* based on $\geq$. In the inference rules, the quasi-order $\geq^*_{\text{lpo}}$ with strict part $>^*_{\text{lpo}}$ and equivalence part $=^*_{\text{lpo}}$ is the lexicographic extension of $\geq_{\text{lpo}}$.

$$(1) \quad \frac{\text{var}(x)}{x =_{\text{lpo}} x}$$

$$(2) \quad \frac{s_i \geq_{\text{lpo}} g(t_1, \ldots, t_n) \text{ for some } i}{f(s_1, \ldots, s_m) >_{\text{lpo}} g(t_1, \ldots, t_n)}$$

$$(3) \quad \frac{f > g \quad f(s_1, \ldots, s_m) >_{\text{lpo}} t_i \text{ for all } i}{f(s_1, \ldots, s_m) >_{\text{lpo}} g(t_1, \ldots, t_n)}$$

$$(4) \quad \frac{f = g \quad (s_1, \ldots, s_m) =^*_{\text{lpo}} (t_1, \ldots, t_n)}{f(s_1, \ldots, s_m) =_{\text{lpo}} g(t_1, \ldots, t_n)}$$

$$(5) \quad \frac{f = g \quad (s_1, \ldots, s_m) >^*_{\text{lpo}} (t_1, \ldots, t_n) \quad f(s_1, \ldots, s_m) >_{\text{lpo}} t_i \text{ for } 2 \leq i \leq n}{f(s_1, \ldots, s_m) >_{\text{lpo}} g(t_1, \ldots, t_n)}$$

Note that in inference rule (1) var(x) is true if and only if x is a variable. Also note that in inference rule (5) it is not necessary to test $f(s_1, \ldots, s_m) >_{\text{lpo}} t_1$ since $(s_1, \ldots, s_m) >^*_{\text{lpo}} (t_1, \ldots, t_n)$ implies $s_1 \geq_{\text{lpo}} t_1$ which, since $f(s_1, \ldots, s_m) >_{\text{lpo}} s_1$, implies $f(s_1, \ldots, s_m) >_{\text{lpo}} t_1$. ∎

Theorem 2.32 All lexicographic path orderings are quasi-simplification orderings.

Proof: Proved in Kamin and Lévy[55]. Also stated without proof in Dershowitz[27]. ∎

Theorem 2.33 The strict part of all lexicographic path orderings have the full invariance property.

Proof: By induction on the definition of the lexicographic path ordering. ∎

Further discussion on the termination of term rewriting systems can be found in Dershowitz[27], Dershowitz and Jouannaud[29], and Plaisted[82].

2.8.5 Ordered Rewriting

There are some equations, such as commutativity ($x * y = y * x$), which cannot be ordered under any termination ordering. This limits the effectiveness of term rewriting. Ordered rewriting is an extension to term rewriting which allows unorderable equations to be use in simplifying terms under certain conditions while preserving termination.

Definition 2.123 Let E be a set of equations and $>$ a partial ordering on terms. The pair $(E, >)$, denoted by $E_>$, forms an *ordered rewriting system.* We refer to each equation $\{ s = t \}$ in E as a *rewrite rule* of $E_>$. ∎

Definition 2.124 Let $E_>$ be an ordered rewriting system and s and t terms. We say s *rewrites* to t under $E_>$, denoted by $s \rightarrow_{E_>} t$, if there exists an equation $\{ l \doteq r \}$ in E, a substitution θ, and a position p such that $s[l\theta]_p$, t is $s[l\theta \rightarrow r\theta]_p$ (recall that $s[l\theta \rightarrow r\theta]_p$ denotes the term obtained by replacing the term $l\theta$ at position p by the term $r\theta$), and $s > t$. We say $\rightarrow_{E_>}$ is the *rewrite relation* of $E_>$. We refer to $l\theta$ as the *redex* of $s \rightarrow_{E_>} t$. ∎

Definition 2.125 Let $E_>$ be an ordered rewriting system and s and t terms. We say s is $E_>$*-reducible* if there exists a term u such that $s \rightarrow_{E_>} u$; otherwise, we say s is $E_>$*-irreducible.* We say t is an $E_>$*-normal form* of s if $s \rightarrow^*_{E_>} t$ and t is $E_>$-irreducible. ∎

Definition 2.126 Let $E_>$ be an ordered rewriting system. We say $E_>$ is *reduced* if for all equations $\{ l = r \}$ in $E_>$, l is $E'_>$-irreducible and r is $E_>$-irreducible where E' is $E - \{ l = r \}$. ∎

Definition 2.127 Let $E_>$ be an ordered rewriting system. We say $E_>$ is *canonical* if $E_>$ is reduced, terminating, and confluent. We say $E_>$ is *ground canonical* if $E_>$ is reduced, terminating, and ground confluent. ∎

Ordered rewriting extends to clauses and sets of clauses in a straightforward manner.

Definition 2.128 Let $E_>$ be an ordered rewriting system and C and D clauses. We say C *rewrites* to D under $E_>$, denoted by $C \rightarrow_{E_>} D$, if there exist terms s and t and position p such that $s \rightarrow_{E_>} t$, $C[s]_p$, and D is $C[s \rightarrow t]_p$ (recall that $C[s \rightarrow t]_p$ denotes the clause obtained by replacing the term s at position p by the term t). ∎

Definition 2.129 Let $E_>$ be an ordered rewriting system and C and D clauses. We say C is *$E_>$-reducible* if there exists a clause C' such that $C \rightarrow_{E_>} C'$; otherwise, we say C is *$E_>$-irreducible*. We say D is an *$E_>$-normal form* of C if $C \rightarrow^*_{E_>} D$ and D is $E_>$-irreducible. ∎

Definition 2.130 Let $E_>$ be an ordered rewriting system and S and T sets of clauses. We say S *rewrites* to T under $E_>$, denoted by $S \rightarrow_{E_>} T$, if there exist clauses C and D such that $C \rightarrow_{E_>} D$, S contains C, and T is $S - \{ C \} \cup \{ D \}$. ∎

Definition 2.131 Let $E_>$ be an ordered rewriting system and S and T sets of clauses. We say S is *$E_>$-reducible* if there exists a set S' of clauses such that $S \rightarrow_{E_>} S'$; otherwise, we say S is *$E_>$-irreducible*. We say T is an *$E_>$-normal form* of S if $S \rightarrow^*_{E_>} T$ and T is $E_>$-irreducible. ∎

Theorem 2.34 Let $E_>$ be an ordered rewriting system. If $>$ is a termination ordering, then $\rightarrow_{E_>}$ is terminating.

Proof: Directly from the definitions of termination ordering and ordered rewriting. ∎

Theorem 2.35* Let $E_>$ be an ordered rewriting system and C and D clauses such that $C \rightarrow^*_{E_>} D$. Then $E \cup \{ C \} \models_= D$.

Proof: Directly from the definitions of ordered rewriting and E-logical consequence. ∎

2.8.6 Completion

A canonical term rewriting system R acts as a decision procedure for $\mathrm{Th}(R)$. To decide if an equation $\{ s = t \}$ is in $\mathrm{Th}(R)$, one simply rewrites s and t to their R-normal forms s' and t', respectively, and then tests if s' and t' are identical. Thus, given a set of equations, it would very useful to find an equivalent canonical term rewriting system, if one exists. Knuth and Bendix[58] developed a procedure, known as *Knuth-Bendix completion*, which, given a finite set of equations and a termination ordering, attempts to find an equivalent canonical term rewriting system ordered by the termination ordering.

We present Knuth-Bendix completion as a set of inference rules, following somewhat the notation of Dershowitz[28]. We begin by giving general definitions for completion sequence and completion procedure.

Definition 2.132 Let E be a set of equations and R a term rewriting system. We say the pair (E, R) is a *completion pair*. We refer to a sequence $(E_0, R_0), (E_1, R_1), \ldots$ of completion pairs as a *completion sequence*. ∎

Definition 2.133 Let (E, R) be a completion pair. We say (E, R) is *finite* if both E and R are finite. ∎

Definition 2.134 Let (E, R) be a completion pair. The *rewrite relation* of (E, R), denoted by $\to_{(E,R)}$, is $\to_E \cup \to_R$. ∎

Definition 2.135 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a completion sequence. The set E^∞ of *persisting equations* of $(E_0, R_0), (E_1, R_1), \ldots$ is $\bigcup_{i\geq 0} \bigcap_{j \geq i} E_j$. Similarly, the set E^∞ of *persisting rewrite rules* of $(E_0, R_0), (E_1, R_1), \ldots$ is $\bigcup_{i\geq 0} \bigcap_{j \geq i} R_j$. ∎

Definition 2.136 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a completion sequence. We say (E_0, R_0), $(E_1, R_1), \ldots$ is *sound* if for all $i \geq 1$, $s \leftrightarrow^*_{(E_i,R_i)} t$ iff $s \leftrightarrow^*_{(E_{i-1},R_{i-1})} t$ for all terms s and t. ∎

Definition 2.137 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a completion sequence. We say (E_0, R_0), $(E_1, R_1), \ldots$ is *successful* if E^∞ is empty and R^∞ is canonical. ∎

Definition 2.138 A *completion procedure* is an algorithm, possibly non-terminating, which takes a finite set of equations E_0, a termination ordering $>$, and a finite term rewriting system R_0 ordered by $>$ as input and outputs a completion sequence $(E_0, R_0), (E_1, R_1), \ldots$ such that for all $i \geq 1$, R_i is ordered by $>$. ∎

Definition 2.139 Let C be a completion procedure. We say C is *sound* if all completion sequences generated by C are sound. ∎

Definition 2.140 Let C be a completion procedure. We say C is *correct* if for all completion sequences generated by C, E^∞ is empty implies R^∞ is canonical. ∎

Definition 2.141 Let C be a completion procedure, E_0 a set of equations, $>$ a termination ordering, R_0 a term rewriting system ordered by $>$, $(E_0, R_0), (E_1, R_1), \ldots$ the completion sequence generated by running C on $(E_0, >, R_0)$, and E^∞ the set of persisting equations of $(E_0, R_0), (E_1, R_1), \ldots$. We say C *fails* for $(E_0, >, R_0)$ if E^∞ is not empty. ∎

We now present Knuth-Bendix completion.

Definition 2.142 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a completion sequence, $>$ be a termination ordering ordering R_0, and $\rhd$[9] a well-founded partial ordering on terms. We say $(E_0, R_0), (E_1, R_1), \ldots$ is a *Knuth-Bendix completion sequence* if for all $i \geq 0$, (E_{i+1}, R_{i+1}) is the result of a single application of one of the following inference rules to (E_i, R_i):

Delete: $$\frac{(E \cup \{\{ s = s \}\}, R)}{(E, R)}$$

Compose: $$\frac{(E, R \cup \{ s \to t \})}{(E, R \cup \{ s \to u \})} \quad \text{if } t \to_R u$$

Simplify: $$\frac{(E \cup \{\{ s \doteq t \}\}, R)}{(E \cup \{\{ s \doteq u \}\}, R)} \quad \text{if } t \to_R u$$

Orient: $$\frac{(E \cup \{\{ s \doteq t \}\}, R)}{(E, R \cup \{ s \to t \})} \quad \text{if } s > t$$

Collapse: $$\frac{(E, R \cup \{ s \to t \})}{(E \cup \{\{ u = t \}\}, R)} \quad \text{if } s \to_R u \text{ by rewrite rule } l \to r \text{ and } s \rhd l$$

Deduce: $$\frac{(E, R)}{(E \cup \{\{ s = t \}\}, R)} \quad \text{if there exists a term } u \text{ such that } u \to_R s \text{ and } u \to_R t$$

We say $>$ is the *termination ordering* of $(E_0, R_0), (E_1, R_1), \ldots$ and $\rhd$ is the *specialization ordering* of $(E_0, R_0), (E_1, R_1), \ldots$. We refer to the inference rules as the *Knuth-Bendix completion inference rules.* ∎

Definition 2.143 Let C be completion procedure. We say C is a *Knuth-Bendix completion procedure* if all completion sequences generated by C are Knuth-Bendix completion sequences. ∎

Theorem 2.36 (Soundness of the Knuth-Bendix Completion Inference Rules) Let (E, R) and (E', R') be completion pairs such that (E', R') is the result of a single application of a Knuth-Bendix completion inference rule to (E, R). Then $E \cup R \models_= E' \cup (R')^=$.

Proof: Directly from each Knuth-Bendix completion inference rule. ∎

[9] $\rhd$ is usually taken to be the *specialization ordering*, where $s \rhd t$ iff some subterm of s is an instance of t and no subterm of t is no an instance of s.

Theorem 2.37 (Soundness of Knuth-Bendix Completion) All Knuth-Bendix completion procedures are sound.

Proof: Directly from the Soundness of the Knuth-Bendix Completion Inference Rules (Theorem 2.36). ∎

Given a finite completion pair, there exists only a finite number of applications of the first five Knuth-Bendix completion inference rules. However, the deduce inference rule can be applied in infinitely many ways. This raises the question on how does one control application of the deduce inference rule. Knuth and Bendix[58] showed that application of the deduce inference rule can be restricted to *critical pairs*, which represent selected "overlaps" of the left hand sides of rewrite rules.

Definition 2.144 Let R be a term rewriting system, $l_1 \to r_1$ and $l_2 \to r_2$ rewrites rules, possibly the same rewrite rewrite rule, in R (with variables renamed so that $l_1 \to r_1$ and $l_2 \to r_2$ have no variables in common), and t a subterm of l_1 at position p. We say $(r_1\theta, l_1\theta[l_2\theta \to r_2\theta]_p)$ (recall that $l_1\theta[l_2\theta \to r_2\theta]_p$ denotes the term obtained by replacing $l_2\theta$ at position p by $r_2\theta$) is a *critical pair* of R if θ is a most general unifier of l_2 and t. We use $\mathrm{CP}(R)$ to denote the set of all critical pairs of R. ∎

Theorem 2.38 (Critical Pair Theorem) Let R be a terminating term rewriting system. Then R is confluent iff for all critical pairs (s, t) in $\mathrm{CP}(R)$, some arbitrarily chosen R-normal forms of s and t are identical.

Proof: Proved in Plaisted[82]. ∎

Definition 2.145 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a Knuth-Bendix completion sequence. We say $(E_0, R_0), (E_1, R_1), \ldots$ is *fair* if

(a) E^∞ is empty.
(b) R^∞ is reduced.
(c) $\mathrm{CP}(R^\infty) \subseteq \bigcup E_i$.

Definition 2.146 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a Knuth-Bendix completion sequence. We say $(E_0, R_0), (E_1, R_1), \ldots$ *fails* if $(E_0, R_0), (E_1, R_1), \ldots$ is not the prefix of some fair Knuth-Bendix completion sequence. ∎

Definition 2.147 Let C be a Knuth-Bendix completion procedure. We say C is *fair* if C generates only fair or failing Knuth-Bendix completion sequences. ■

Theorem 2.39 (Correctness of Knuth-Bendix Completion) All fair Knuth-Bendix completion procedures are correct.

Proof: Proved in Huet[51]. ■

Knuth-Bendix completion suffers from two major problems. Firstly, Knuth-Bendix completion requires that a termination ordering be specified. For a given set of equations, Knuth-Bendix completion may fail for some termination orderings while succeeding for others. Finding an appropriate termination ordering can be quite difficult in some cases.

One approach to finding an appropriate termination ordering is the *extended termination ordering method*, in which the termination ordering is dynamically extended as the Knuth-Bendix completion procedure attempts to order equations. For example, the lexicographic path ordering can be extended in terms of its quasi-precedence ordering. In this case, a Knuth-Bendix completion procedure starts with the lexicographic path ordering based on the quasi-precedence ordering which orders no function symbols. Each time the Knuth-Bendix completion procedure attempts to order an equation, the quasi-precedence ordering is extended, if necessary and possible, with new orderings on function symbols so that the equation can be ordered. Note that the new lexicographic path ordering is a superset of the previous lexicographic path ordering; thus, all previous orderings of equations remain valid under the new lexicographic path ordering.

It is also possible to apply backtracking to the extended termination ordering method. When an attempt to order an equation fails, the Knuth-Bendix completion procedure backtracks to a point where the termination ordering was extended. It then proceeds by chosing a different extension to the termination ordering. In this way, the Knuth-Bendix completion procedure fails only if there exists no extension of the termination ordering which generates an equivalent canonical term rewriting system.

The second problem with Knuth-Bendix completion is that it may fail to find an equivalent canonical term rewriting system even though one exists[10]. Bachmair, Dershowitz,

[10]For an example of where Knuth-Bendix completion fails to find an existing equivalent canonical term rewriting system, we refer the reader to Section 7 of Bachmair, Dershowitz, and Plaisted[5].

and Plaisted[5] developed *unfailing completion*, an extension to Knuth-Bendix completion, to address this problem. Unfailing completion, in addition to always finding an equivalent term rewriting system if one exists, also always finds, given a termination ordering which is total on ground terms, an equivalent ground canonical ordered rewriting system, even when an equivalent canonical term rewriting system does not exist.

In presenting unfailing completion, we follow the notation of Bachmair, Dershowitz, and Plaisted[5], which presents unfailing completion as an extension to the Knuth-Bendix completion inference rules. Note that unfailing completion can also be described in terms of sequences of ordered rewriting systems rather than sequences of completion pairs.

Definition 2.148 Let $(E_0, R_0), (E_1, R_1), \ldots$ be a completion sequence, $>$ be a termination ordering ordering R_0, and $\rhd$[11] a well-founded partial ordering on terms. We say $(E_0, R_0), (E_1, R_1), \ldots$ is an *unfailing completion sequence* if for all $i \geq 0$, (E_{i+1}, R_{i+1}) is the result of a single application of one of the following inference rules to (E_i, R_i):

Delete: $\dfrac{(E \cup \{\,\{\, s = s \,\}\,\}, R)}{(E, R)}$

Compose: $\dfrac{(E, R \cup \{\, s \to t \,\})}{(E, R \cup \{\, s \to u \,\})}$ if $t \to_R u$

Compose$_2$: $\dfrac{(E, R \cup \{\, s \to t \,\})}{(E, R \cup \{\, s \to u \,\})}$ if $t \to_{E_>} u$

Simplify: $\dfrac{(E \cup \{\,\{\, s \doteq t \,\}\,\}, R)}{(E \cup \{\,\{\, s \doteq u \,\}\,\}, R)}$ if $t \to_R u$

Simplify$_2$: $\dfrac{(E \cup \{\,\{\, s \doteq t \,\}\,\}, R)}{(E \cup \{\,\{\, s \doteq u \,\}\,\}, R)}$ if $t \to_{E_>} u$ by rewrite rule $l \to r$ and $s \rhd l$

Orient: $\dfrac{(E \cup \{\,\{\, s \doteq t \,\}\,\}, R)}{(E, R \cup \{\, s \to t \,\})}$ if $s > t$

Collapse: $\dfrac{(E, R \cup \{\, s \to t \,\})}{(E \cup \{\,\{\, u = t \,\}\,\}, R)}$ if $s \to_R u$ by rewrite rule $l \to r$ and $s \rhd l$

---

[11] As with Knuth-Bendix completion, $\rhd$ is usually taken to be the specialization ordering.

Collapse$_2$:	$\dfrac{(E, R \cup \{ s \to t \})}{(E \cup \{ \{ u = t \} \}, R)}$	if $s \to_{E_>} u$ by rewrite rule $l \to r$ and $s \vartriangleright l$
Deduce$_2$:	$\dfrac{(E, R)}{(E \cup \{ \{ s = t \} \}, R)}$	if there exists a term u such that $s \leftrightarrow_{(E \cup R^=)_>} u \leftrightarrow_{(E \cup R^=)_>} t$

We say $>$ is the *termination ordering* of $(E_0, R_0), (E_1, R_1), \ldots$ and $\vartriangleright$ is the *specialization ordering* of $(E_0, R_0), (E_1, R_1), \ldots$. We refer to the inference rules as the *unfailing completion inference rules.* ■

Definition 2.149 Let C be completion procedure. We say C is an *unfailing completion procedure* if all completion sequences generated by C are unfailing completion sequences. ■

Theorem 2.40* (Soundness of the Unfailing Completion Inference Rules) Let (E, R) and (E', R') be completion pairs such that (E', R') is the result of a single application of an unfailing completion inference rule to (E, R). Then $E \cup R \models_= E' \cup (R')^=$.

Proof: Directly from each unfailing completion inference rules. ■

Theorem 2.41 (Soundness of Unfailing Completion) All unfailing completion procedures are sound.

Proof: Directly from the Soundness of the Unfailing Completion Inference Rules (Theorem 2.40). ■

As with the deduce inference rule of Knuth-Bendix completion, the major question for unfailing completion is how to control application of the deduce$_2$ inference rule. Bachmair, Dershowitz, and Plaisted[5] showed, in manner similar to that of Knuth and Bendix, that application of the deduce$_2$ inference rule can be restricted to *extended critical pairs*, which represent selected "overlaps" of equations.

Definition 2.150 Let $>$ be a partial ordering on terms, E a set of equations, $\{ l_1 \doteq r_1 \}$ and $\{ l_2 \doteq r_2 \}$ equations, possibly the same equation, in E (with variables renamed so that $\{ l_1 \doteq r_1 \}$ and $\{ l_2 \doteq r_2 \}$ have no variables in common), and t a subterm of l_1 at position p. We say $(r_1\theta, l_1\theta[l_2\theta \to r_2\theta]_p)$ (recall that $l_1\theta[l_2\theta \to r_2\theta]_p$ denotes the term obtained by replacing $l_2\theta$ at position p by $r_2\theta$) is an *extended critical pair* of E with respect to $>$ if θ is a most general unifier of l_2 and t, $r_1\theta \not> l_1\theta$, and $r_2\theta \not> l_2\theta$. We use $\mathrm{EP}_>(E)$ to denote the set of all extended critical pairs on E with respect to $>$. ■

Definition 2.151 Let $(E_0, R_0), (E_1, R_1), \ldots$ be an unfailing completion sequence and $>$ the termination ordering of $(E_0, R_0), (E_1, R_1), \ldots$. We say $(E_0, R_0), (E_1, R_1), \ldots$ is *fair* for $>$ if $\mathrm{EP}_>(E^\infty \cup (R^\infty)^=) \subseteq \bigcup E_i$. ■

Definition 2.152 Let C be an unfailing completion procedure. We say C is *fair* if for all termination ordering $>$, C generates only fair unfailing completion sequences. ■

Theorem 2.42 (Correctness of Unfailing Completion) Let C be a fair unfailing completion procedure. Then for all sets of equations E_0, termination orderings $>$, and term rewriting systems R_0 ordered by $>$, running C on $(E_0, >, R_0)$ generates a completion sequence such that $(E^\infty \cup (R^\infty)^=)_>$ is ground canonical.

Proof: Proved in Bachmair, Dershowitz, and Plaisted[5]. ■

Corollary 2.43* Let $(E_0, R_0), (E_1, R_1), \ldots$ be a fair unfailing completion sequence and $>$ the termination ordering of $(E_0, R_0), (E_1, R_1), \ldots$. Then $(E^\infty \cup (R^\infty)^=)_>$ is ground canonical.

Proof: Directly from Correctness of Unfailing Completion (Theorem 2.42). ■

Theorem 2.44 (Completeness of Unfailing Completion) Let C be a fair unfailing completion procedure. Then for all sets of equations E, termination orderings $>$ contained within a partial ordering on terms complete for E, and canonical term rewriting systems R ordered by $>$ and equivalent to E, running C on $(E, >, \{\,\})$ generates a completion sequence where E^∞ is empty and R and R^∞ are identical up to the renaming of variables.

Proof: Proved in Bachmair, Dershowitz, and Plaisted[5]. ■

2.8.7 Specialized Unification

Another technique for addressing the term rewriting limitation of unorderable equations is specialized unification. Specialized unification is similar to unification. However, instead of trying to make a set of terms identical, specialized unification attempts to make a set of terms equal with respect to set of equations. For example, the terms $f(x, a)$ and $f(y, b)$ cannot be unified. However if f is commutative, the substitution $\{\, x \mapsto b, y \mapsto a \,\}$ makes $f(x, a)$ and $f(y, b)$ equal. Specialized unification algorithms have been developed for a number of equational theories. In particular, much study has been done on associative-commutative unification. Stickel[91] presents one such associative-commutative unification algorithm. Siekmann[89,90] presents a good survey on special specialized unification.

2.9 Hyper-Linking

The Skolem-Herbrand-Gödel theorem provides an attractive way to solve the theorem proving problem via instance-based proof procedures. The main issue with instance-based proof procedures is which ground instances should be generated. One way of generating ground instances is to simply enumerate them. This approach was used by Gilmore[39]. The problem with this approach is the exponential number of ground instances required to prove most theorems. In order for an instance-based proof procedure to efficiently prove most theorems, it must restrict the ground instances generated.

The hyper-link operation is a method for generating an instance of a clause from an unsatisfiable set of clauses, which when the instance is added to the set, brings the set closer[12] to the finite unsatisfiable subset of ground instances of the Skolem-Herbrand-Gödel theorem. The hyper-link operation differs from the commonly used resolution inference rule, developed by Robinson[88], in that it does not combine instances of literals from different clauses to form new clauses. In fact, Lee and Plaisted[63] claim that it is this difference which distinguishes hyper-linking from other commonly used automated theorem proving methods.

As shown by Lee and Plaisted[62,63], the hyper-link operation can be used to form an instance-based refutation proof procedure by repeatedly generating clause instances from a given clause set and testing the resulting clause set for propositional unsatisfiability. Such proof procedures are known as *hyper-link proof procedures.* Lee and Plaisted[62,63] showed that if the hyper-link operation is applied appropriately the resulting hyper-link proof procedure is sound and logically complete.

We now define the hyper-link operation.

[12]For a precise definition of closer, see Lee[62].

Definition 2.153 Let S be a set of clauses and $C = \{ L_1, \ldots, L_n \}$ a clause in S. The *hyper-link operation* maps C into the instance $C\theta$ where θ is a most general unifier such that for each L_i there exists a literal M_i (with variables renamed so that L_i and M_i have no variables in common) in some clause of S for which $L_i\theta$ and $M_i\theta$ are contradictory. We refer to C as the *nucleus* of the hyper-link of C in S, θ as the *substitution* of the hyper-link of C in S, $C\theta$, denoted by $\mathrm{HL}_S(C)$, as the *instance* of the hyper-link of C in S, each M_i as an *electron* of the hyper-link of C in S, and each pair (L_i, M_i) as a *link* of the hyper-link of C in S. A link (L_i, M_i) is a *unit link* if M_i occurs in a unit clause of S. ∎

The following example illustrates the hyper-link operation. Let S be the following set of clauses:

C_1: $\{ r(a, w) \}$
C_2: $\{ p(a), {\sim}r(b, c) \}$
C_3: $\{ {\sim}q(x), r(a, x) \}$
C_4: $\{ {\sim}p(y), {\sim}s(z), {\sim}r(y, z) \}$
C_5: $\{ q(b), {\sim}r(c, b) \}$
C_6: $\{ s(b) \}$

S has five links (written as unordered pairs of literals):

l_1: $(r(a, w), {\sim}r(y, z))$
l_2: $(p(a), {\sim}p(y))$
l_3: $(q(b), {\sim}q(x))$
l_4: $(r(a, x), {\sim}r(y, z))$
l_5: $(s(b), {\sim}s(z))$

These five links are used to generate the following four hyper-link instances:

i_1: $\{ r(a, w) \}$ from nucleus C_1 using link l_1
i_2: $\{ {\sim}q(b), r(a, b) \}$ from nucleus C_3 using links l_3 and l_4
i_3: $\{ {\sim}p(a), {\sim}s(b), {\sim}r(a, b) \}$ from nucleus C_4 using links l_2, l_5, and l_4
i_4: $\{ s(b) \}$ from nucleus C_6 using link l_5

Definition 2.154 Let S be a set of clauses and \$ an individual constant not occurring in S. The *ground* of S, denoted by $\mathrm{Gr}(S)$, is the set of ground clauses obtained by replacing each variable in S by \$. ∎

Definition 2.155 Let S be a set of clauses. We say S is *propositionally unsatisfiable* if $\mathrm{Gr}(S)$ is unsatisfiable; otherwise, we say S is *propositionally satisfiable.* ∎

Theorem 2.45* Let S be a set of clauses. If S is propositionally unsatisfiable, then S is unsatisfiable.

Proof: Directly from the definitions of propositionally unsatisfiability and unsatisfiability. ■

Definition 2.156 Let $S_0, S_1, \ldots$ be a sequence of sets of clauses. We say $S_0, S_1, \ldots$ is a *hyper-link sequence* if for $i \geq 0$, S_{i+1} is $S_i \cup \{C\}$ where C is an instance of a hyper-link in S_i such that S_i does not contain a variant of C. ■

Theorem 2.46* **(Soundness of Hyper-Linking)** Let $S_0, S_1, \ldots$ be a hyper-link sequence. If any S_i in $S_0, S_1, \ldots$, is unsatisfiable, then S_0 is unsatisfiable.

Proof: Directly from the definition of a hyper-link sequence. ■

Definition 2.157 Let $S_0, S_1, \ldots$ be a hyper-link sequence. We say $S_0, S_1, \ldots$ is *fair* if for every S_i in $S_0, S_1, \ldots$ and every instance C of a hyper-link of S_i, there exists an S_j containing a variant of C. ■

Theorem 2.47* **(Completeness of Hyper-Linking)** Let S_0 be an unsatisfiable set of clauses and $S_0, S_1, \ldots$ be a fair hyper-link sequence. Then there exists an S_n in $S_0, S_1, \ldots$, such that S_n is propositionally unsatisfiable.

Proof: Proved in Lee[62]. ■

Lee and Plaisted[62,63] used the hyper-link operation to develop the following hyper-link proof procedure:

1. Set S_0 to the Skolemization of S and $\sim F$.
2. Set i to 0.
3. Determine if S_i is propositionally unsatisfiable. If so, stop and report that F is a logical consequence of S.
4. Generate instances of the clauses in S_i using the hyper-link operation. Let S_i' be the set of instances generated. Set S_{i+1} to $S_i' \cup S_i$.
5. If $S_{i+1} = S_i$, stop and report that F is not a logical consequence of S.
6. Set i to $i + 1$.
7. Go to step 3.

They refer to each iteration of the loop as a hyper-link round. We, therefore, refer to the above hyper-link proof procedure as the *round-by-round hyper-link proof procedure*. This

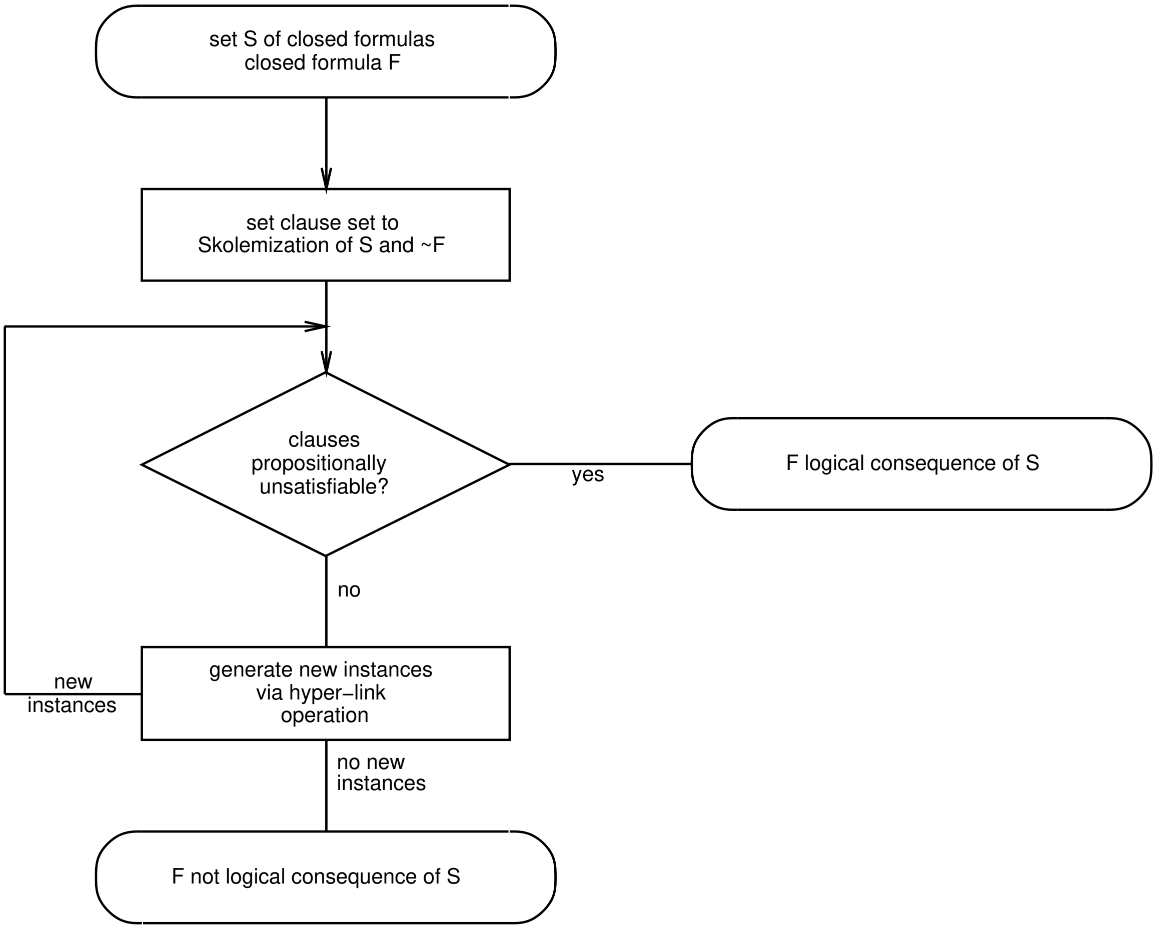

Figure 2.2: The Round-by-Round Hyper-Linking Proof Procedure

Algorithm *Hyper_Link_Clause_Set*(S, I)
Input
S: set of clauses
Output
I: $\{ \mathrm{HL}_S(C) : C \in S \}$

begin
$I := \{ \}$
$E := \{ M : M \text{ is a literal in } S \}$
for each $C \in S$ **do**
Hyper_Link_Clause$(\{ \}, C, E, I')$
$I := I \cup I'$
end

Algorithm *Hyper_Link_Clause*(P, U, E, I)
Input
P: set of previously linked literals
U: set of unlinked literals
E: set of potential electrons
Output
I: set of instances generated by linking the literals in U

begin
if $U = \{ \}$ **then** $I := \{ P \}$
else
choose an L from U
$I := \{ \}$
for each $M \in E$ **do**
if there exists a most general unifier θ such that
$L\theta$ and $M\theta$ are contradictory **then**
Hyper_Link_Clause$(P\theta \cup \{ L\theta \}, U\theta - \{ L\theta \}, E, I')$
$I := I \cup I'$
end

Figure 2.3: Algorithms *Hyper_Link_Clause_Set* and *Hyper_Link_Clause*

procedure is illustrated in Figure 2.2. A recursive algorithm for generating the set of hyper-link instances of a clause set is presented in Figure 2.3. Lee[62] used the round-by-round hyper-link proof procedure as the basis for his automated theorem prover CLIN.

Lee and Plaisted[62,63] developed a number of refinements to the round-by-round hyper-link proof procedure. These include support strategies, tautology deletion, unit simplification, UR resolution, ground unit generation, instance deletion, small proof checking, predicate replacement, sliding priority, and clause splitting.

2.9.1 Support Strategies

Lee and Plaisted[62,63] developed three restrictions to the hyper-link operation which they referred to as *support strategies*. The support strategies restrict the hyper-link operation, thus reducing the number of instances generated. The three support strategies are backward support, forward support, and user support.

In backward support, we say a clause is *backward supported* if it contains only negative literals or if it is an instance of a backward supported hyper-link. We say an electron is *backward supported* if it occurs in a backward supported clause. We say a link is *backward supported* if the literal from the nucleus is negative or the electron is backward supported. An instance of a backward supported hyper-link is an instance of a hyper-link in which all links are backward supported.

Forward support is simply backward support in which the roles of the positive and negative literals have been reversed. We say a clause is *forward supported* if it contains only positive literals or if it is an instance of a forward supported hyper-link. We say an electron is *forward supported* if it occurs in a forward supported clause. We say a link is *forward supported* if the literal from the nucleus is positive or the electron is forward supported. An instance of a forward supported hyper-link is an instance of a hyper-link in which all links are forward supported.

In user support, selected input clauses are marked as user supported. We say a clause is *user supported* if it is a user supported input clause or an instance of a user supported hyper-link. We say an electron is *user supported* if it occurs in a user supported clause. We say a link is *user supported* if the electron is user supported. An instance of a user supported hyper-link is an instance of a hyper-link in which either the nucleus is user supported or at least one link is user supported. We note that user support is less restrictive that backward and forward support in that it only restricts the hyper-link operation on selected input clauses.

Recall the previous example of hyper-linking. Let S be the following set of clauses:

C_1: $\{ r(a, w) \}$
C_2: $\{ p(a), \sim r(b, c) \}$
C_3: $\{ \sim q(x), r(a, x) \}$
C_4: $\{ \sim p(y), \sim s(z), \sim r(y, z) \}$
C_5: $\{ q(b), \sim r(c, b) \}$
C_6: $\{ s(b) \}$

S has five links (written as unordered pairs of literals):

l_1: $(r(a, w), \sim r(y, z))$
l_2: $(p(a), \sim p(y))$
l_3: $(q(b), \sim q(x))$
l_4: $(r(a, x), \sim r(y, z))$
l_5: $(s(b), \sim s(z))$

These five links are used to generate the following four hyper-link instances:

i_1: $\{ r(a, w) \}$ from nucleus C_1 using link l_1
i_2: $\{ \sim q(b), r(a, b) \}$ from nucleus C_3 using links l_3 and l_4
i_3: $\{ \sim p(a), \sim s(b), \sim r(a, b) \}$ from nucleus C_4 using links l_2, l_5, and l_4
i_4: $\{ s(b) \}$ from nucleus C_6 using link l_5

Assuming that S is a set of input clauses (that is, none of the are hyper-link instances), clause C_4 is backward supported, clauses C_1 and C_6 are forward supported, and clauses C_2, C_3, and C_5 are neither backward nor forward supported. The backward supported hyper-link instances are i_1, i_2, i_3, and i_4. The forward supported hyper-link instances are i_1 and i_4. The hyper-link instance i_2 is not forward supported since the link $(q(b), \sim q(x))$ is not forward supported. The link $(q(b), \sim q(x))$ is not forward supported since the electron $q(b)$ does not come from a forward supported clause. The hyper-link instance i_3 is not forward supported since the link $(p(a), \sim p(y))$ is not forward supported. The link $(p(a), \sim p(y))$ is not forward supported since the electron $q(b)$ does not come from a forward supported clause. One should note that as hyper-link instances are add, electrons which were previously not supported may become supported.

Lee and Plaisted[62,63] showed that completeness of hyper-linking is maintained when the hyper-link operation is restricted to only backward or forward supported hyper-linking. They also showed that user supported hyper-linking is complete when the set of input clauses not marked as user supported is satisfiable.

2.9.2 Tautology Deletion

For some theorem proving problems, the hyper-link operation generates a number of instances of the form $\{L\} \cup \{\sim L\} \cup C$. Clauses of this form are always valid, and thus cannot contribute towards the unsatisfiability of a clause set. Such clauses are referred to as *tautologies*.

Definition 2.158 Let C be a clause and L an atom. We say C is *tautology* if C contains both L and $\sim L$. ∎

Theorem 2.48 Let S be a set of clause and C a tautology. Then $S \cup \{C\}$ is unsatisfiable iff S is unsatisfiable.

Proof: Directly from the definitions of tautology and unsatisfiability. ∎

Since tautologies do not contribute towards the unsatisfiability of a clause set, they can be deleted from the clause set using the following inference rule:

$$\textbf{Tautology Deletion}: \quad \frac{S \cup \{\{L\} \cup \{\sim L\} \cup C\}}{S}$$

2.9.3 Unit Simplification

Unit clauses represent unconditional truth, and are, therefore, very useful in eliminating redundant information. Two inference rules capture the power of unit clauses:

$$\textbf{Unit Subsumption}: \quad \frac{S \cup \{\{L\} \cup C\} \cup \{\{M\}\}}{S \cup \{\{M\}\}} \quad \text{where } L \text{ is an instance of } M$$

$$\textbf{Unit Literal Deletion}: \quad \frac{S \cup \{\{L\} \cup C\} \cup \{\{M\}\}}{S \cup \{C\} \cup \{\{M\}\}} \quad \begin{array}{l}\text{where } L \text{ is a contradictory} \\ \text{instance of } M\end{array}$$

The unit subsumption inference rule says that any clause C containing an instance of a unit clause $\{M\}$ can be deleted. In this case, we say that C is *unit subsumed* by $\{M\}$.

The unit literal deletion inference rule says that any contradictory instance of a unit clause can be deleted from any clause. Note that application of the unit literal deletion inference rule may result in the addition of the empty clause to a clause set, which in turn makes the clause set trivially unsatisfiable. Also, application of the unit literal deletion inference rule may result in the addition of a unit clause to a clause set. In this case, the newly added unit clause may be used in further applications of the unit subsumption and unit literal deletion inference rules.

We define the unit literal deletion of a clause to be the result of applying the unit literal deletion inference rule as many times as possible to a clause.

Definition 2.159 Let S be a set of clauses and $\{L_1, \ldots, L_k, L_{k+1}, \ldots, L_n\}$ and $\{L_1, \ldots, L_k\}$ clauses in S. We say $\{L_1, \ldots, L_k\}$ is the *unit literal deletion* of $\{L_1, \ldots, L_k, L_{k+1}, \ldots, L_n\}$ in S if

(a) for $1 \leq i \leq k$, there exists no unit clause $\{M_i\}$ in S such that L_i is an instance of M_i.
(b) for $k+1 \leq i \leq n$, there exists a unit clause $\{M_i\}$ in S such that L_i is an instance of M_i. ∎

Definition 2.160 Let S and S' be sets of clauses. We say S' is the *unit literal deletion* of S if S' is $\{C' : C \in S$ and C' is the unit literal deletion of C in $S\}$. ∎

Lee and Plaisted[62,63] showed that the unit subsumption and unit literal deletion inference rules can be combined with the hyper-link operation to form a more powerful proof procedure. However, one must be careful in deciding when to apply the unit subsumption inference rule. For example, consider the following unsatisfiable clause set: $\{\{p(x,a)\}, \{\sim p(a,x)\}\}$. This set is not propositionally unsatisfiable so at least one application of the hyper-link operation is required. Note that any instance generated by the hyper-link operation can be unit subsumed by its nucleus. Thus, if the unit subsumption inference rule is applied immediately after the hyper-link operation, no proof is ever found. To prevent problems of this type from occurring, one can apply the unit literal deletion inference rule between application of the hyper-link operation to a unit clause and application of the unit subsumption inference rule to the resulting instance.

One should also note that the unit literal deletion inference rule can be combined directly with the hyper-link operation by deleting every literal in the instance of the of hyper-link which resulted from linking a literal in the nucleus with a literal from a unit clause. Such use of the unit literal deletion inference rule is referred to as *embedded unit literal deletion*.

Unit subsumption and unit literal deletion are sound.

Theorem 2.49 (Soundness of Unit Subsumption) For all sets S of clauses, all clauses C, and all literals L and M such that L is an instance of M, $S \cup \{ \{ L \} \cup C \} \cup \{ \{ M \} \} \models S \cup \{ \{ M \} \}$.

Proof: Directly from the definition of unit subsumption. ∎

Theorem 2.50* (Soundness of Unit Literal Deletion) For all sets S of clauses, all clauses C, and all literals L and M such that L is an contradictory instance of M, $S \cup \{ \{ L \} \cup C \} \cup \{ \{ M \} \} \models S \cup \{ C \} \cup \{ \{ M \} \}$.

Proof: Directly from the definition of unit literal deletion. ∎

2.9.4 UR Resolution

Lee and Plaisted[62,63] noted that the unit resulting (UR) resolution strategy (see Wos, Overbeek, Lusk, and Boyle[97]) was consistent with the hyper-linking philosophy of not combining instances of literals to form new clauses. They also noted that the UR strategy could be simulated by restricting the hyper-link operation in a manner similar to that of the support strategies.

UR resolution is an operation similar to the hyper-link operation in which all but one literal is linked with literals from unit clauses and the remaining literal is not linked. By embedded unit literal deletion, all of the linked literals are deleted, resulting in a unit instance. While UR resolution is not, in general, complete by itself, it is complete for sets of Horn clauses. Lee[62] showed that when combined with the backward, forward, or user supported hyper-link operation or with small proof checking (described in a following section), UR resolution can be used to prove some moderately difficult theorems.

2.9.5 Ground Unit Generation

Lee and Plaisted[62,63] developed an inference rule, which they referred to as *ground unit generation*, for replacing selected non-unit clauses with ground unit clauses. Let C be a clause containing a ground literal L such that $\{L\}$ is the result of an application of UR resolution to C. By unit subsumption, C can be deleted by $\{L\}$. The combination of UR resolution and unit subsumption, in which a clause containing a ground literal is replaced with the unit clause containing only the ground literal, is *ground unit generation.*

Ground Unit Generation: $\dfrac{S \cup \{C\}}{S \cup \{\{L\}\}}$ where L is a ground literal in S and $\{L\}$ is the result of an application of UR resolution to C

2.9.6 Instance Deletion

With hyper-linking, one cannot delete instances of clauses, as is common with many resolution based theorem provers. However, it is possible to generalize the unit subsumption inference rule, although Lee[62] did not implement this generalization in CLIN. The generalization to unit subsumption is expressed by the following inference rule:

Proper Subsumption: $\dfrac{S \cup \{C\} \cup \{D\}}{S \cup \{D\}}$ where some proper subset of C is an instance of D

The proper subsumption inference rule says that any clause C can be deleted if some proper subset of C is an instance of some other clause D. In this case, we say that C is *properly subsumed* by D.

Furthermore, when using the hyper-link operation with embedded unit literal deletion, one can delete instances of clauses for clause sets provable by UR resolution. In particular, this means that when using the hyper-link operation with embedded unit literal deletion on sets of Horn clauses, one can delete instances of clauses. Lee[62] referred to the deletion of clauses as *instance deletion*.

2.9.7 Small Proof Checking

For most theorems, the round-by-round hyper-link proof procedure suffers, as do most breadth-first proof procedures, from an extremely fast growing search space. In particular, the average number of clauses in the clause set after hyper-link round n is given by the recurrence[13]:

$$\begin{aligned} T(0) &= k \\ T(n) &= bT(n-1) + T(n-1) \\ &= (b+1)T(n-1) \end{aligned}$$

where k is the number of input clauses and b is the average number of new hyper-link instances generated per clause per round. Solving the recurrence gives the average number of clauses in the the clause set after hyper-link round n as $k(b+1)^n$.

One technique developed by Lee and Plaisted[62,63] to address this problem is *small proof checking*. The idea of small proof checking is to search for small proofs after performing the propositional unsatisfiability test in the hope of eliminating the need for further hyper-link rounds.

Note that the elimination of one or two hyper-link rounds can result in substantial savings. For example, consider a theorem for which $k = 10$ and $b = 3$. Say that the round-by-round hyper-link proof procedure finds a proof during hyper-link round 6. Thus, the clause set contains 40,960 clauses when the proof is found. If the proof could be found during hyper-link round 5, the clause set would only contain 10,240 clauses. If, even better, the proof could be found during hyper-link round 4, the clause set would only contain only 2,560 clauses, which is only one-sixteenth the number of clauses needed to find the proof during hyper-link round 6.

A small proof is simply an application of the hyper-link operation in which all of the electrons come from unit clauses. By embedded unit literal deletion, the instance of this hyper-link is the empty clause. Thus, any clause set having a small proof is unsatisfiable. Lee and Plaisted[62,63] extended the definition of a small proof by allowing electrons to

[13] Our analysis differs from that of Lee[62] in that Lee made the simplifying assumption that a clause contributes new hyper-link instances to the clause set only in the hyper-link round immediately after it is added to the clauses set. We note, instead, that a clause may contribute hyper-link instances for many hyper-link rounds, as new electrons are generated during each hyper-link round. In Lee's analysis, the average number of new hyper-link instances generated during hyper-link round n is kb^n where k is the number of input clauses and b is the average number of new hyper-link instances generated per clause during the hyper-link round immediately after the clause is added to the clause set. This gives the average number of clauses in the clause set after hyper-link round n as $k(\frac{b^{n+1}-1}{b-1})$.

come from the result of a single application of UR resolution, as well as from unit clauses. They defined the size of a small proof to the be number of links performed in the applications of UR resolution used to generate the electrons. They referred to a small proof in which all of the electrons come directly from unit clauses as a *simple small proof* and a small proof in which one or more electron comes from a single application of UR resolution as a *complicated small proof*. Note that finding a simple small proof reduces the number of hyper-link rounds required to find the proof by one, and finding a complicated small proof reduces the number of hyper-link rounds required to find the proof by two, assuming, of course, use of the general or UR resolution hyper-link operation. Finding a complicated small proof when the backward or forward supported hyper-link operation is used may reduce the number of hyper-link rounds required to find the proof by more than two, as more than one hyper-link round may be required to simulate the UR-resolution steps of the complicated small proof.

Lee and Plaisted[62,63] further showed that it is only necessary to consider input clauses as nuclei for small proofs. They also showed that, when unit literal deletion is applied, only unit clauses or non-unit clauses containing no ground literals need be considered as sources for electrons in small proofs.

Lee[62] implemented small proof checking in CLIN by adding a check for small proofs after the propositional unsatisfiability test. When searching for complicated small proofs, he used iterative deepening on the small proof size. That is, CLIN first searches for a small proof of size 0, then a small proof of size 1, and so on. The time spent searching for a small proof is controlled in two ways: (1) the CLIN user may specify a maximum bound on the size of the small proof and (2) the time spent searching for a small proof during a hyper-link round is limited to a percentage of the time spent generating hyper-link instances during the hyper-link round. One interesting consequence of the second technique is that improving the efficiency in generating hyper-link instances may result in longer proof times for some theorems, as CLIN may no longer find a small proof within the reduced time bound.

2.9.8 Predicate Replacement

In some theories, such as von Neumann-Bernays-Gödel set theory, many predicates are simply defined in terms of other predicates. One intuitive approach for reducing the search space is to expand predicate definitions within clauses. However, early work by Plaisted and Greenbaum[85] and Plaisted and Potter[87] showed that simple replacement of a clause C by a clause C' in which one or more predicate definitions are expanded could, in some cases, make a proof much harder to find. Rather, they noted that it is better to add additional clauses, referred to as *replacement rules*, which allow for the fast derivation of the clause C' from the clause C.

Lee and Plaisted[62,63] developed an operation, similar to the hyper-link operation, in which selected clauses are used as replacement rules. They referred to this operation as *predicate replacement.* We first give the definition of a replacement rule.

Definition 2.161 Let $\{ L_1, \ldots, L_n \}$ be a clause and $\{ L_1, \ldots, L_k \}$ a non-empty, proper subset of $\{ L_1, \ldots, L_n \}$. We use $M_1, \ldots, M_k \rightarrow L_{k+1}, \ldots, L_n$, where each M_i is the negation of L_i, to denote the *replacement rule* of $\{ L_1, \ldots, L_n \}$ with *distinguished literals* $L_1, \ldots, L_k$. We refer to a replacement rule having only one distinguished literal as a *definitional replacement rule.* ∎

The predicate replacement operation is basically the hyper-link operation applied to a predicate replacement rule in which only the distinguished literals are linked. We restrict the substitution in electrons so the replacement only occurs for those literals having the proper term structure. We also restrict the literals used as electrons to prevent more than one replacement being applied simultaneously to a literal.

Definition 2.162 Let S be a set of clauses, C a clause, and $R = M_1, \ldots, M_k \rightarrow L_{k+1}, \ldots, L_n$ a replacement rule of C. The *predicate replacement operation* maps C into the instance $C\theta$

where θ is a most general unifier such that for each M_i there exists a literal N_i (with variables renamed so the M_i and N_i have no variables in common) in some clause of S for which

(a) $M_i\theta$ and $N_i\theta$ are identical.
(b) θ maps every variable in N_i to either a variable or a constant.
(c) N_i does not occur only as the instance of a distinguished literal in an instance of predicate replacement in S.

We refer to C as the *nucleus* of a predicate replacement of R in S, θ as the *substitution* of the predicate replacement of R in S, $C\theta$ as the *instance* of the predicate replacement of R in S, each N_i as an *electron* of the predicate replacement of R in S, and each pair (L_i, N_i) as a *link* of the predicate replacement of R in S. ∎

In general, repeated application of the predicate replacement operation is not terminating. However, Lee and Plaisted[62,63] showed that if the replacement rules obey certain ordering properties, application of predicate replacement operation can be guaranteed to always terminate.

Definition 2.163 Let $\geq$ be a quasi-ordering on literals. We say $\geq$ is a *replacement ordering* if

(a) for all literals L, $\{ M : L \geq M \}$ is finite.
(b) for all literals L and M and substitutions θ, $L \geq M$ implies $L\theta \geq M\theta$. ∎

Definition 2.164 Let R be a set of replacement rules and $\geq$ a replacement ordering. We say R is *ordered* by $\geq$ if for every replacement rule r in R and every non-distinguished literal L in r, there exists a distinguished literal M in r such that $M \geq L$.

Definition 2.165 Let R be a set of replacement rules and $S_0, S_1, \ldots$ a sequence of sets of clauses. We say that $S_0, S_1, \ldots$ is an *R-replacement sequence* if for $i \geq 0$, S_{i+1} is $S_i \cup \{ C \}$ where C is an instance of the predicate replacement of r in S_i for some replacement rule r in R such that S_i does not contain a variant of C. ∎

Definition 2.166 Let R be a set of replacement rules. We say R is *terminating* if there is no infinite R-replacement sequence starting from a finite set of clauses; otherwise, we say R is *non-terminating.* ∎

Theorem 2.51 Let R be a finite set of replacement rules. If R is ordered by a replacement ordering, then R is terminating.

Proof: Proved in Lee[62]. ∎

Lee[62] used the predicate replacement operation to prove many difficult theorems in set theory and temporal logic, thus illustrating its potential power. However, predicate replacement has one major limitation in that it requires substantial knowledge of the input clauses in order to identify the appropriate replacement rules. In many cases, the necessary replacement rules are not intuitive or natural and are difficult to identify. Chu[18], in his development and implementation of semantic hyper-linking, defined two types of replacement rules, *natural replacement rules* and *minimal replacement rules*, which he generated automatically.

Definition 2.167 Let R be a replacement rule. We say R is *natural* if

(a) R has the form $M_1, \ldots, M_k \rightarrow L$.
(b) Every variable in L occurs in some M_i. ∎

Definition 2.168 Let $R = M_1, \ldots, M_k \rightarrow L_{k+1}, \ldots, L_n$ be a replacement rule. We say R is *minimal* if $\{ M_1, \ldots, M_k \}$ is a minimal subset of $\{ M_1, \ldots, M_k, L_{k+1}, \ldots, L_n \}$ such that every variable in R occurs in some M_i. ∎

One should note that Chu's usage of predicate replacement differs from that of Lee's. While Lee used predicate replacement to simply generate new clauses, Chu used predicate replacement to find contradictions of models. It remains an open question whether automatic generation of natural and minimal replacement rules is applicable to hyper-linking proof procedures.

2.9.9 Sliding Priority

Another common technique for controlling the size of the search space is the *size bound method.* In the size bound method, a maximum size bound is specified, and all clauses with sizes exceeding the specified size bound are deleted. An iterative deepening technique is sometimes used in conjunction with the size bound method; if a proof is not found at the initial size bound, the size bound is increased, and the proof procedure is run again. This process is repeated until the proof is found.

One major problem with the size bound method is choosing the appropriate initial size bound. For some theorems, a small size bound may be necessary to find the proof, while for very similar theorems, a fairly large size bound is necessary.

Lee and Plaisted[62,63] developed a new technique, *sliding priority*, to address the problem of determining the initial size bound. Instead of specifying the initial size bound, a work bound b is given. A work size w is maintained as the sum of the sizes of the clauses in the clause set. The size bound s is set to the maximum clause size for which $w \leq b$, where, of course, clauses with sizes larger than s are deleted. As new clauses are generated, the work bound w increases. Eventually, w becomes larger than b. When this happens, the size bound s is adjusted to the maximum clause size for which $w \leq b$, where all clauses with sizes larger than s are deleted. In this manner the size bound "slides" downward, based upon the amount of work done. An iterative deepening technique can also be used with sliding priority; if a proof is not found for the initial work bound, the work bound is is increased, and the proof procedure run again.

One should note that the sliding priority technique described above is not restricted to just hyper-link proof procedures; it has general applicability to other proof procedures as well. One should also note the the sliding priority technique is fairly general in that different measures can be used for clause and work sizes.

Lee[62] implemented sliding priority in CLIN. The initial work bound is set automatically, based on the input clause set. When a proof is not found for a particular work bound, the work bound is doubled and the round-by-round hyper-link proof procedure is run again. Lee referred to each running of the round-by-round hyper-link proof procedure as a *hyper-link session*.

Lee and Plaisted[62,63] studied two measures for clause size: (1) clause size as the sum of the sizes of the literals in the clause, where the size of a literal is the number of predicate symbols, function symbols, and variables in the literal and (2) clause size as the maximum size of the literals in the clause, again where the size of a literal is the number of predicate symbols, function symbols, and variables in the literal. They found that the sum of literals sizes works better for some theorems, while for other theorems the maximum literal size works better.

2.9.10 Clause Splitting

The number of hyper-link instances for a given nucleus is exponential in the number literals; thus, hyper-linking usually performs better on clauses with fewer literals. In some cases, a clause with a large number of literals can clutter the hyper-link search space with many unnecessary hyper-link instances. Lee[62] presented *clause splitting* as a method for reducing the number of literals in a clause.

Clause splitting works by splitting a clause with a large number literals into two clauses, each having fewer literals than the original. Given a clause $\{ L_1, \ldots, L_n \}$ with $n \geq 3$ clause splitting replaces $\{ L_1, \ldots, L_n \}$ with the clauses $\{ L_1, \ldots, L_i, p(x_1, \ldots, x_m) \}$ and $\{ \sim p(x_1, \ldots, x_m), L_{i+1}, \ldots, L_n \}$ where p is a new predicate symbol and $x_1, \ldots, x_m$ are the variables common between L_1, $\ldots$, L_i and L_{i+1}, $\ldots$, L_n. Note that the clause splitting method can be applied recursively to split a clause with a large number of literals into several clauses each containing only a small number of literals.

Chapter 3

Adding Equality Support to Round-by-Round Hyper-Linking

Our first attempt in adding equality support to hyper-linking was to extend the round-by-round hyper-link proof procedure of Lee and Plaisted[62,63]. We implemented our ideas by extending Lee's automated theorem prover CLIN. Our modifications to CLIN can be grouped into two broad classes, *general enhancements*, which apply not only to equality theorems, but also improve the efficiency of CLIN on non-equality theorems as well, and *equality enhancements*, which are in direct support of equality. We first present the general enhancements and then the equality enhancements. We next discuss changes made to the high-level structure of CLIN. We conclude the chapter with a discussion on why we feel the round-by-round hyper-link proof procedure is not well suited for proving equality theorems.

3.1 General Enhancements to Round-by-Round Hyper-Linking

The general enhancements to round-by-round hyper-linking are deletion of selected partial hyper-links, elimination of unnecessary hyper-linking, automated clause splitting, a modification to sliding priority, fixed priority, and a modification to the hyper-link session end criteria.

3.1.1 Deletion of Selected Partial Hyper-Links

In CLIN the hyper-link operation is performed by choosing a nucleus and then linking each literal in the nucleus one at a time. Further hyper-link instances of the nucleus are then generated via backtracking. We refer to a nucleus in which some of the literals have been linked as a *partial hyper-link*.

Definition 3.1 Let S be a set of clauses and $C = \{ L_1, \ldots, L_n \}$ a clause in S. Let θ be a most general unifier such that for some subset $\{ L'_1, \ldots, \mathrm{L}'_{n'} \}$ of literals in C, there exists a literal M_i in some clause of S for which $L'_i\theta$ and $M_i\theta$ are contradictory. We refer to $C\theta$ as a *partial hyper-link* of C in S, C as the *nucleus* of the partial hyper-link $C\theta$, each M_i as an *electron* of the partial hyper-link $C\theta$, and each pair (L'_i, M_i) as a *link* of the partial hyper-link $C\theta$. We refer to the literal $L_i\theta$ as a *linked literal* in the partial hyper-link $C\theta$ if $L_i \in \{ L'_1, \ldots, L'_{n'} \}$ and as an *unlinked literal* in the partial hyper-link $C\theta$ if $C\theta$ if $L_i \notin \{ L'_1, \ldots, L'_{n'} \}$ ∎

We use an underscore to indicate that a literal in a partial hyper-link is linked. For example, $\{ p(a), q(b), r(x) \}$ is an unlinked partial hyper-link, $\{ \underline{p(a)}, q(b), \underline{r(x)} \}$ is a partially linked partial hyper-link in which the literals $p(a)$ and $r(x)$ are linked, and $\{ \underline{p(a)}, \underline{q(b)}, \underline{r(x)} \}$ is a fully linked partial hyper-link.

Definition 3.2 Let S be a set of clauses, $C = \{ L_1, \ldots, L_n \}$ a clause in S, and $C\theta$ a partial hyper-link of C. We say $C\theta$ is *fully linked* if every literal $L_i\theta$ is linked; otherwise, we say $C\theta$ is *partially linked*. We also say $C\theta$ is *unlinked* if no literal $L_i\theta$ is linked. ∎

Definition 3.3 Let C and D be partial hyper-links with unlinked literals $\{ L_1, \ldots, L_n \}$ and $\{ M_1, \ldots, M_n \}$, respectively. We say C and D are *duplicates*, denoted by $C \simeq D$ if

(a) $C \approx D$.
(b) $\{ L_1, \ldots, L_n \} \approx \{ M_1, \ldots, M_n \}$. ∎

We note that in CLIN a hyper-link instance C is not added to (or is deleted from) the clause set if

(1) C is a variant of some clause in the clause set.
(2) C is a tautology.
(3) The size of C size exceeds the current size bound.
(4) C is subsumed by a unit clause.

Let C be a partially linked partial hyper-link. Assume that C is a duplicate of a partial hyper-link D previously generated during the same hyper-link round as C. For every hyper-instance C' coming from C, there is a corresponding hyper-link instance D' coming from D such that $C' \approx D'$. Thus, there is no need to continue linking C. We refer to the test for duplicate partial hyper-links as *duplicate partial hyper-link deletion.*

Let C be a partially linked partial hyper-link. Assume that C is a tautology. Then every hyper-instance C' coming from C is also a tautology. Thus, there is no need to continue linking C. We refer to the test for partial hyper-link tautologies as *early tautology deletion.*

Let C be a partially linked partial hyper-link. Assume that the size of C exceeds the current size bound. Then the size of every hyper-link instance C' coming from C also exceeds the current size bound, with two exceptions: (1) a literal in C' is deleted by embedded unit literal deletion and (2) C' has fewer literals than C because one or more literals became identical as unlinked literals in C were linked. In the first case, we can delete all unlinked literals which might possibly be linked with an electron from a unit clause before computing the size of C. In the second case, we can detect those pairs of unlinked literals which may possibly become identical and delete the larger literal of each pair before computing the size of C. We refer to the test for partial hyper-links whose size exceeds the current size bound as *early priority deletion* and early priority deletion with "unsafe" literals deleted as *safe early priority deletion.*

Let C be a partially linked partial hyper-link. Assume C is subsumed by a unit clause D. Then every hyper-link instance C' coming from C is also unit subsumed by D. Thus, there is no need to continue linking C. We refer to the unit subsumption test of partial hyper-links as *early unit subsumption.*

3.1.2 Elimination of Unnecessary Hyper-Linking

The round-by-round hyper-linking proof procedure is inefficient in that subsequent hyper-link rounds unnecessarily redo the hyper-linking of previous hyper-link rounds. We have eliminated this unnecessary hyper-linking by only performing hyper-link operations in which either the nucleus or as least one electron was not available for hyper-linking in the previous hyper-link round.

3.1.3 Automated Clause Splitting

The number of hyper-link instances generated by the hyper-link operation is exponential in the number of literals in the nucleus. One particular source of long clauses is Brand's E-modification. We have automated clause splitting to address the problem with long clauses.

3.1.4 Modification to Sliding Priority

Sliding priority is a technique for reducing the search space by dynamically lowering the size bound based on the amount of work performed. We noticed that sliding priority in CLIN does not, for theorems with large search spaces, lower the size bound quickly enough.

We address this problem by changing the definition of the sliding priority work bound w from the sum of the sizes of the clauses in the clause set to the sum of the sizes of each generated partial hyper-link. This, in conjunction with early priority deletion, offers, in general, a better measure of the work performed.

3.1.5 Fixed Priority

One problem with sliding priority is that it is possible for an improvement which reduces the size of the search space to actually increase the time it takes to find a proof. When the size of the search space is reduced, the priority bound decreases at a slower rate, thus allowing larger hyper-link instances to be generated. In some cases, the generation of these larger hyper-link instances takes more time than that saved by the reduction in the size of the search space. This makes the evaluation of implementation changes difficult.

One alternative to sliding priority, which does not exhibit this problem, is the size bound method. We refer to our implementation of the size bound method as *fixed priority.*

Although results indicate that sliding priority performs better than fixed priority, fixed priority has proven useful in the evaluation of implementation changes.

3.1.6 Modification to Hyper-Link Session Termination

Under sliding priority, when a proof is not found for a given work bound and one or more clauses were not added to (or were deleted from) the clause set because their sizes exceeded the size bound, CLIN doubles the sliding priority work bound and repeats the round-by-round hyper-link proof procedure. Each running of the round-by-round hyper-link proof procedure is referred to as a hyper-link session. CLIN determines when to start a new hyper-link session by comparing the clause set before a hyper-link round with the clause set after the hyper-link round. If the clause sets differ, an additional hyper-link round is performed.

In determining if any new clauses are generated during a hyper-link round, CLIN considers a clause which changes support state as a new clause. For example, if a clause C becomes forward supported during a hyper-link round, CLIN treats C as a new clause and proceeds with another hyper-link round. This is necessary when forward supported hyper-linking is used; however, if CLIN is not performing forward supported hyper-linking and the only "new" clauses generated during a hyper-linking round are clauses which became forward supported, CLIN performs an unnecessary hyper-link round.

CLIN allows support strategies to be varied between hyper-link rounds. For example, the user may choose to have the support strategy alternate between one round of forward supported hyper-linking and one round of backward supported hyper-linking. Consider the case where a forward supported hyper-link round is performed in which no new clauses are generated, no existing clause becomes forward supported, and one or more existing clauses become backward supported. Since one or more existing clauses became supported, CLIN performs a backward supported hyper-link round. Assume that during the backward supported hyper-link round, no new clauses are generated, no existing clause becomes forward supported, and that one or more existing clauses become backward supported. Since an existing clause changed support state, CLIN performs a forward supported hyper-link round. Note, however, that this forward supported hyper-link round is unnecessary, since no new clauses have been generated and no existing clause has become forward supported since the last forwarded supported hyper-link round. Assume that during this forward supported hyper-link round no new clauses are generated and no existing clause becomes supported. A backward supported hyper-link round should now be performed as one or more clauses became backward supported since the last backward supported hyper-link round was performed. However, CLIN erroneously ends the current hyper-link session since during the last hyper-link round no new clauses were generated and no existing clause became supported.

We address the above problems by independently testing each specific support strategy hyper-link round for new clauses. For example, when determining if a forward support hyper-link round needs to be performed, we compare the current clause set to the clause set prior to the last forward supported hyper-link round. We also perform similar tests before performing backward supported and UR resolution hyper-link rounds. We end a hyper-link session only when hyper-linking is complete for all specified support strategies.

3.2 Equality Enhancements to Round-by-Round Hyper-Linking

The equality enhancements are Brand's E-modification, term rewriting, ordered rewriting, critical pair generation, automated group detection, and orienting equations.

3.2.1 Brand's E-Modification

We have added the ability for CLIN to automatically apply Brand's E-modification to the set of input clauses and to automatically add the reflexivity and transitivity axioms.

We note that for clauses containing deeply nested terms, the E-modification can result in clauses with many literals. Thus, we have experimented with variations of Brand's E-modification. For example, we offer an equality transformation which differs from Brand's E-modification in that it only pulls out common subterms once. Brand's E-modification of $\{ p(a,a) \}$ is $\{ a \neq x, a \neq y, p(x,y) \}$, while our variation results in $\{ a \neq x, p(x,x) \}$. Pulling out common subterms only once reduces, in many cases, the number of literals in the transformed clause.

Another variation to Brand's E-modification which we tried was to not pull-out constants. In this case, the equality transformation of $\{ p(a,a) \}$ is simply $\{ p(a,a) \}$.

3.2.2 Term Rewriting

We have added term rewriting to CLIN by automatically generating rewrite rules from equations. Equations are ordered into rewrite rules using the lexicographic path ordering.

We perform term rewriting during hyper-linking as hyper-link instances are generated, ordering new equations into rewrite rules as they occur. For efficiency, we do not rewrite existing clauses as new rewrite rules are generated. Instead, we have added a separate rewrite step after hyper-linking which rewrites all clauses to normal form[14]. During this rewrite step, equations are first fully rewritten, taking care not to use a rewrite rule to rewrite the left hand side of its corresponding equation. Fully rewriting the equations results in a reduced term rewriting system, which is then used to rewrite the remaining non-equation clauses. We have also investigated not rewriting clauses during the hyper-linking step of a hyper-link round, rather only rewriting clauses during the separate rewrite step.

For efficiency, we use a modified outermost term rewriting strategy. Our term rewriting strategy consist of first fully rewriting a term's outermost function and then recursively rewriting the resulting term's arguments. This process is repeated until no changes occurs.

Rewriting large terms to normal form may involve many hundreds, possibly thousands, of rewrite steps. Many automated theorem provers, such as OTTER and RRL, address this problem by placing a bound on the number of rewrite steps allowed in a single rewrite. We introduce a new approach. Instead of using a bound on the number of rewrite steps, we check the size of the term after the application of certain rewrite rules. If the size of the term exceeds the size bound, we stop rewriting the term.

Since it is relatively expensive to compute the size of a term, we restrict the size test to specially marked rewrite rules. A rewrite rule is marked if it contains a variable which

[14]Note that the term rewriting system generated by ordering equations is not necessarily confluent. Therefore, a given clause may have more than one normal form. The rewrite step simply rewrites each clause to some normal form.

occurs more times on the rewrite rule's right hand side than on its left hand side. For example, $j(0, x) \rightarrow x$ is not marked, while $f(x, j(y, z)) \rightarrow j(f(x, y), f(x, z))$ is.

In general, term rewriting does not preserve unsatisfiability; it only preserves E-unsatisfiability. For example, the clause set $\{ \{ f(a) = a \}, \{ p(f(a)) \}, \{ \sim p(f(x)) \} \}$ is obviously unsatisfiable, since $\{ p(f(a)) \}$ and $\{ \sim p(f(x)) \}$ link. Applying term rewriting to this set results in $\{ \{ f(a) = a \}, \{ p(a) \}, \{ \sim p(f(x)) \} \}$, which is no longer unsatisfiable. Another way of looking at this is that term rewriting may "break" the hyper-link proof in that a pair of literals which link before term rewriting may no longer link afterwards. For example, consider a link between $p(f(x))$ and $\sim p(f(0))$. Rewriting $\sim p(f(0))$ with $f(0) = 0$ gives $\sim p(0)$, which no longer links with $p(f(x))$.

One way of overcoming this problem is to apply the equality transformation after each rewrite step. This, however, is fairly inefficient, especially as larger hyper-link instances are generated. We believe that restricting term rewriting so that it does not rewrite the outermost function of the arguments to $=$ or $\neq$ may offer an alternative solution to this problem; however, we have been unable to show completeness when restricting term rewriting in this way.

3.2.3 Ordered Rewriting

We have extended term rewriting with ordered rewriting, using the lexicographic path ordering as the partial ordering on terms. Note that each potential rewrite step requires a lexicographic path ordering comparison, which can be quite expensive to compute. We, therefore, order equations into rewrite rule whenever possible. In rewriting a term, we first rewrite the term fully using rewrite rules, for which no lexicographic path ordering comparison is necessary, and then rewrite the resulting term once using unordered equations. This process is repeated until no change occurs. In rewriting with either rewrite rules or unordered equations, we use our modified outermost rewriting strategy to determine which redex to rewrite.

3.2.4 Critical Pair Generation

Brand's E-modification suffers from a major problem in that it can result in clauses containing many literals. These long clauses are not well suited for hyper-linking.

As an alternative to Brand's E-modification, we have enabled CLIN to generate critical pairs and extended critical pairs. Note that when generating critical pairs or extended critical pairs, we add the reflexivity axiom to the set of input clauses. While the generation of critical pairs or extended critical pairs is not logically complete in general, it should prove useful for those theorems involving only equations and inequations.

3.2.5 Automated Group Detection

For many equational systems there exist equivalent canonical term rewriting systems. Also, many equational systems have properties which can be simulated by predicate replacement. Therefore, it would be useful to quickly detected these equational systems. The equations corresponding to the canonical term rewriting system could then be added to the clause set, eliminating the need to generate them via the hyper-linking. Also, predicate replacement rules simulating properties of the equational system could be used. We have implemented automatic detection for group theory as described below.

Knuth and Bendix[58] give the following ten rule canonical term rewriting system for groups:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
e \cdot x & \rightarrow & x & e^{-1} & \rightarrow & e \\
x^{-1} \cdot x & \rightarrow & e & {x^{-1}}^{-1} & \rightarrow & x \\
(x \cdot y) \cdot z & \rightarrow & x \cdot (y \cdot z) & x \cdot x^{-1} & \rightarrow & e \\
x^{-1} \cdot (x \cdot y) & \rightarrow & y & x \cdot (x^{-1} \cdot y) & \rightarrow & y \\
x \cdot e & \rightarrow & x & (x \cdot y)^{-1} & \rightarrow & y^{-1} \cdot x^{-1}
\end{array}$$

Major properties of groups which prove useful in simplifying terms are the cancellation laws, uniqueness of inverse, and uniqueness of identity. The following six replacement rules provide a convenient method for representing these properties:

$$\begin{array}{rclcrcl}
x \cdot y & = & x \cdot z & \to & y & = & z \\
y \cdot x & = & z \cdot x & \to & y & = & z \\
x \cdot y & = & x & \to & y & = & e \\
y \cdot x & = & x & \to & y & = & e \\
x \cdot y & = & e & \to & y & = & x^{-1} \\
y \cdot x & = & e & \to & y & = & x^{-1}
\end{array}$$

We detect groups by looking for the associativity, left inverse, and left identity axioms

$$\begin{array}{rcl}
(x \cdot y) \cdot z & = & x \cdot (y \cdot z) \\
x^{-1} \cdot x & = & e \\
e \cdot x & = & x
\end{array}$$

or the associativity, right inverse, and right identity axioms

$$\begin{array}{rcl}
(x \cdot y) \cdot z & = & x \cdot (y \cdot z) \\
x \cdot x^{-1} & = & e \\
x \cdot e & = & x
\end{array}$$

We allow the binary operation, inverse operation, and identity element to be arbitrary terms containing two, one, and zero distinct variables, respectively. For example, given the axioms

$$\begin{array}{rcl}
f(f(x,x,y), f(x,x,y), z) & = & f(x,x,f(y,y,z)) \\
f(g(x,x,x), g(x,x,x), x) & = & h(a,a,a) \\
f(h(a,a,a), h(a,a,a), x) & = & x
\end{array}$$

we detect the group

$$\begin{array}{ll}
\textbf{group operation}: & f(x,x,y) \\
\textbf{inverse operation}: & g(x,x,x) \\
\textbf{identity element}: & h(a,a,a)
\end{array}$$

Once a group is detected, we assert the ten rewrite rules, their associated equations, and the six replacement rules. One exception to this is when we detect a second group for a given binary operation. In this case, we assert the the following equations and their corresponding rewrite rules

$$\begin{aligned} g(x) &= g'(x) \\ e &= e' \end{aligned}$$

where $g(x)$ and e are the inverse operation and identity element of the first group and $g'(x)$ and e' are the inverse operation and identity element of the second group.

3.2.6 Orienting Equations

The major problem in proving equality theorems is an extremely large search space. One reason for the large search space is the symmetry of equality. We have partially addressed this problem by orienting equations so that the literals $s = t$ and $s \neq t$ where $t >_{\text{lpo}} s$ are replaced by $t = s$ and $t \neq s$, respectively.

3.3 High-Level Structure of CLIN

We have added many new functions to CLIN in support of equality. In adding these new functions, we have significantly modified the high-level structure of CLIN. The new high-level structure not only helps with equality theorems, but also improves the efficiency of CLIN on non-equality theorems as well. The new high-level structure of CLIN is illustrated in Figure 3.1. Figure 3.2 presents clause generation in detail. Each module in clause generation is user selectable, as well as is the equality transformation module.

As an optimization, the clause generation loop is terminated early when it is possible to determine that no new clauses can be generated. For example, consider the case where, during an iteration of clause generation, unit simplification generates new clauses and no new clauses are generated by the remaining modules. The clause generation loop is terminated during the next iteration after tautology deletion since the clause set has not changed since the last time unit simplification was run (tautology deletion generates no new clauses).

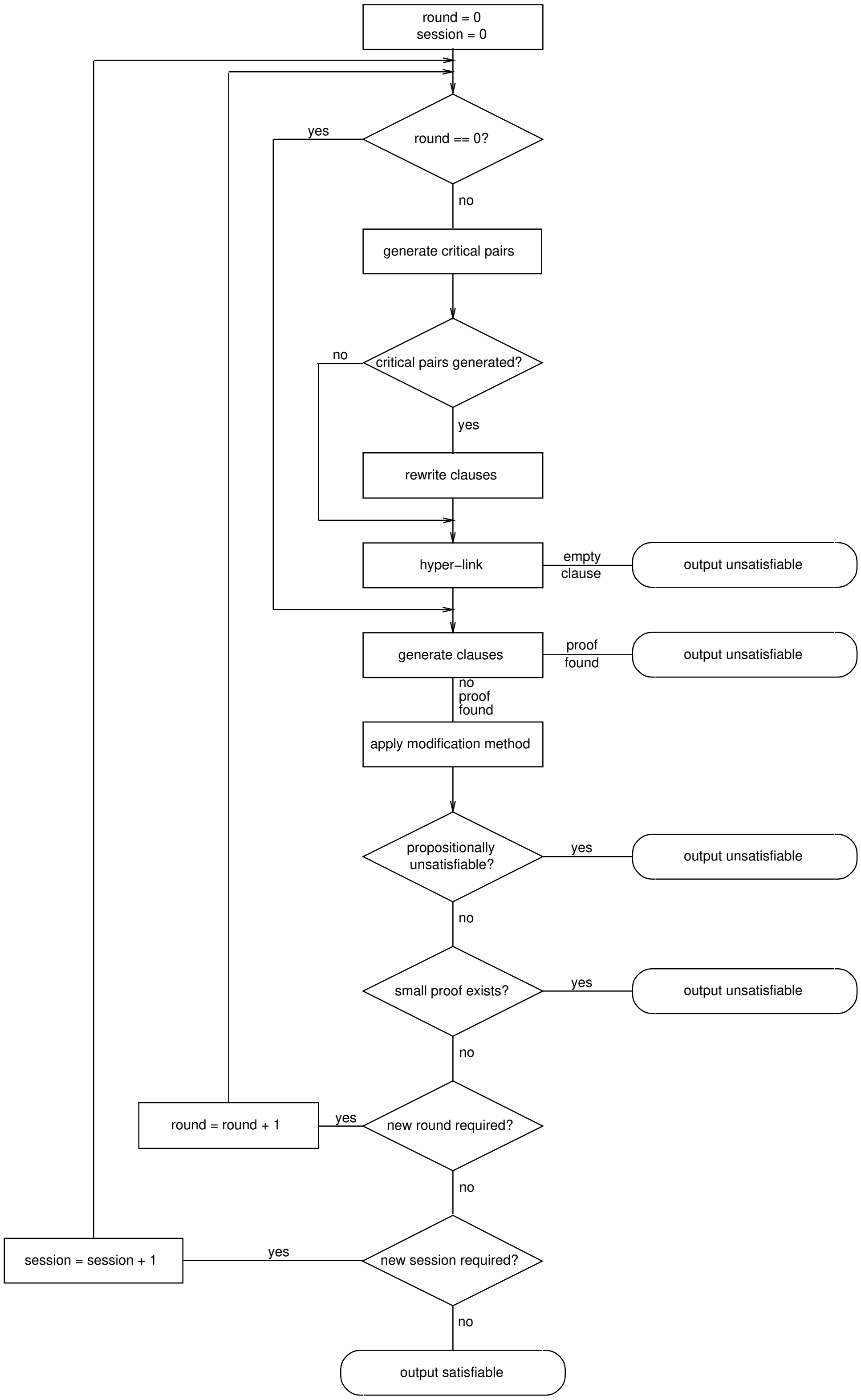

Figure 3.1: The High-Level Structure of CLIN

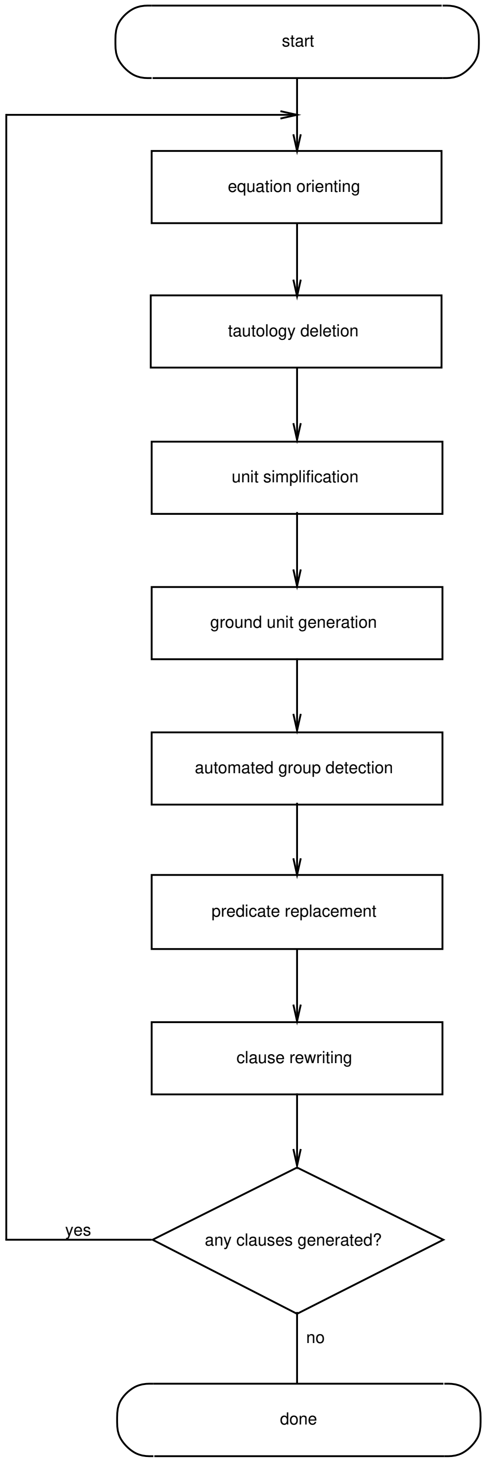

Figure 3.2: Clause Generation Structure

3.4 Results

Lusk and Overbeek[67] present the following six benchmark equality theorems, which are ordered in terms of increasing difficulty:

Problem 1: In a group, if $x^2 = e$ for all x in the group, then the group is Abelian (for all x and y, $xy = yx$).
Problem 2: In a group, $(x^{-1})^{-1} = x$ for all x in the group.
Problem 3: In a ring, if $x^2 = x$ for all x in the ring, then $xy = yx$ for all x, y in the ring.
Problem 4: In a group, if $x^3 = e$ for all x in the group, then the commutator $h(h(x, y), y) = e$ for all x and y. The commutator $h(x, y)$ is defined as $xyx^{-1}y^{-1}$.
Problem 5: In a ternary Boolean algebra with the third axiom[15] removed, it is true that $f(x, g(x), y) = y$ for all x and y.
Problem 6: In a ring, if $x^3 = x$ for all x in the ring, then $xy = yx$ for all x and y in the ring.

We have run both our extended CLIN and OTTER 3.0.2, a resolution based theorem prover developed and implemented by McCune[70], on all six theorems. Some results are given in Table 3.1 where the times are CPU seconds on a DECsystem 5500 with 32MB of memory. OTTER 3.0.2 was run with the "knuth_bendix" flag set. Extended CLIN was run with forward support, instance deletion, clauses sized as the sum of literals sizes, sliding priority (except for problem 4), equality transformation on the input clauses with the untransformed clauses kept, restricted ordered rewriting, equation orienting, and automated group detected. Note that without automated group detection the Extended CLIN times for problems 1 through 4 were significantly slower.

3.5 Conclusion

The times obtained by extended CLIN are fairly slow when compared to OTTER. After much investigation and experimentation, we have concluded that the combination of Brand's E-modification and term rewriting is not an appropriate method for adding equality support to hyper-linking. We base our conclusions on a number of reasons:

[15] The third axiom which is removed is $f(x, y, g(y)) = x$. Lusk and Overbeek also add, as a lemma which can be proven from the remaining axioms, $f(x, y, x) = x$.

Problem	Extended CLIN	OTTER 3.0.2
1	0.95	0.14
2	0.78	0.10
3	119.90	0.98
4	81.12	1.31
5	1,141.68	14.15
6	————	————

Table 3.1: Extended CLIN Results

- Brand's E-modification transforms a clause by pulling out subterms into new literals, thus increasing the number of literals in the clause. This is not well suited for hyper-linking as the number of hyper-link instances of a clause is exponential in the number of literals in the clause.

- Term rewriting does not preserve unsatisfiability; it only preserves E-unsatisfiability. We believe that restricting term rewriting so that it does not rewrite the outermost function of the arguments to $=$ or $\neq$ may address this problem; however, we have been unable to show completeness when restricting term rewriting in this way. Also, in addition to being unable to show completeness, the restriction to term rewriting limits the effectiveness of term rewriting in reducing the size of the search space.

- The round-by-round hyper-link proof procedure suffers, as do most breadth-first proof procedures, from very large search spaces. This is especially true for equality theorems. Lee and Plaisted[62,63] developed sliding priority as a method for controlling the size of the search space. Sliding priority works fairly well for non-equality theorems. However, it no longer performs as well once term rewriting is added. There are two reasons why sliding priority does not work well with term rewriting:

  1. The sum of the sizes of the clauses in the clause set is no longer a good measure of the work performed, as term rewriting, in most cases, rewrites large clauses into small clauses. This means that the work performed rewriting a clause usually decreases the measure of work. It may be possible to modify the work measure to account for the work performed in rewriting clauses; however, we were unable to find a satisfactory measure of work.

2. Sliding priority suffers from the inefficiency of adding large clauses are added to the clause set and then later deleting them. This inefficiency becomes far more significant when term rewriting is added, as term rewriting and the lexicographic path ordering are applied to large clauses which are later deleted.

We now believe that unfailing completion is a much better method for adding equality support to hyper-linking. However, in adding unfailing completion to CLIN, we found that the generation of extended critical pairs suffers greatly from the breadth-first inefficiency of the round-by-round hyper-link proof procedure. In particular, we noted that for many theorems, crucial extended critical pairs are generated only after 10 or more rounds. We feel that a depth-first hyper-link proof procedure is necessary in to order to add efficient equality support to hyper-linking.

Chapter 4

Smallest Instance First Hyper-Linking

In the previous chapter we concluded that the round-by-round hyper-link proof procedure of Lee is not suitable for proving equality theorems. We have developed an alternative class of depth-first hyper-link proof procedures, which we believe to be better suited for proving theorems with large search spaces. These proof procedures, which we refer to as the *smallest instance first hyper-link proof procedures*, are based on the idea that it is better to generate smaller hyper-link instances first.

We first present the class of smallest instance first hyper-link proof procedures and show that they are sound and logically complete. We next discuss methods for choosing the order in which literals in a partial hyper-link are linked. In order for smallest instance first hyper-linking to be a viable alternative to round-by-round hyper-linking, it is necessary to support, where appropriate, the refinements Lee added to round-by-round hyper-linking. Thus, we next discuss how Lee's refinements to round-by-round hyper-linking can be added to smallest instance first hyper-linking. These refinements include support strategies, tautology deletion, unit subsumption, unit literal deletion, UR resolution, ground unit generation, proper subsumption, small proof checking, predicate replacement, sliding priority, and automated clause splitting. We conclude this chapter by discussing two additional enhancements in support of equality, Brand's E-modification and automated group detection. Based on our experience in adding equality support to round-by-round hyper-linking, these two enhancements should improve the efficiency of smallest instance first hyper-linking on equality theorems.

In Chapter 6, we describe a preliminary implementation of smallest instance first hyper-linking. This preliminary implementation only includes some of the refinements to smallest instance first hyper-linking described in this chapter. Of those refinements which are included, some, such as unit subsumption and unit literal deletion, are only partially implemented. For each refinement, we briefly describe how it is included in the preliminary implementation.

4.1 Smallest Instance First Hyper-Link Proof Procedures

Smallest instance first hyper-link proof procedures are refutation, instance-based proof procedures. A smallest instance first hyper-link proof procedure attempts to show that a closed formula F is a logical consequence of a finite set S of closed formulas by first Skolemizing S and $\sim F$ into a set of clauses. It then attempts to show that the set of clauses is unsatisfiable by treating the set of clauses as a set of unlinked partial hyper-links and generating new partial hyper-links by repeatedly linking the smallest linkable partial hyper-link. Whenever a fully linked partial hyper-link is generated, its corresponding clause is added to the clause set, and the clause set is tested for propositional unsatisfiability. If the clause set is determined to be propositionally unsatisfiable, the procedure stops and reports that F is a logical consequence of S. If the set of linkable partial hyper-links becomes empty, the procedure stops and reports that F is not a logical consequence of S.

Note that there are many smallest instance first hyper-link proof procedures, each differing in the quasi-ordering used to determine the order in which linkable partial hyper-links are linked. This quasi-ordering also determines the order in which hyper-link instances are generated.

In general, it is necessary to generate all hyper-link instances of a set of clauses in order to find the smallest hyper-link instance. This can be quite inefficient. However, by requiring that the quasi-ordering defining the smallest hyper-link instance have certain properties, we can significantly reduce the amount of work required to find the smallest hyper-link instance.

Definition 4.1 Let $\geq$ be a quasi-ordering on partial hyper-links with strict part $>$ and equivalence part $=$. We say $\geq$ is a *hyper-link ordering* if

(a) $\geq$ is total.
(b) $>$ is well-founded.
(c) for all partial hyper-links C and D, $C = D$ iff $C \simeq D$ (that is, the hyper-link ordering treats two partial hyper-links as equal if and only if they are duplicates).
(d) for all partial hyper-links C and proper instances $C\theta$ of C, $C\theta > C$. ∎

We note that since partial hyper-links are simply clauses in which selected literals are marked as linked, a hyper-link ordering defines a total quasi-ordering on clauses whose strict part is well-founded and whose equivalence part treats variant clauses as equal.

A smallest instance first hyper-link proof procedure attempts to show that a closed formula F is a logical consequence of a finite set S of closed formulas as follows:

1. Set S_0 to the Skolemization of S and $\sim F$.
2. Set P to the set of unlinked partial hyper-links corresponding to the clauses in S_0.
3. Mark each partial hyper-link in P as linkable.
4. Set i to 0.
5. Determine if S_i is propositionally unsatisfiable. If so, stop and report that F is a logical consequence of S.
6. Set C to the smallest linkable partial hyper-link in P. If every partial hyper-link in P is unlinkable, stop and report that F is not a logical consequence of S.
7. If C is fully linked, do the following:
   a) Mark C as unlinkable.
   b) If C is in S_i, go to step 6.
   c) Set S_{i+1} to $S_i \cup \{C\}$.
   d) If C contains a literal not in some clause of S_i, mark all partially linked partial hyper-links in P as linkable.
   e) Add the unlinked partial hyper-link corresponding to C, marked as linkable, to P.
   f) Set i to $i+1$.
   g) Go to step 5.

8. Else (C is partially linked), do the following:
   a) Choose an unlinked literal L in C and a literal M in a clause in S_i such that L and M link with most general unifier θ and $C\theta$ is not in P. If no such literals L and M exist, do the following:
      i) Mark C as unlinkable.
      ii) Go to step 6.
   b) Add $C\theta$ to P.
   c) Go to step 6.

This process is illustrated in Figure 4.1.

Definition 4.2 Let P be an algorithm, possibly non-terminating, implementing the process presented in Figure 4.1 using the hyper-link ordering $\geq$ when choosing the smallest linkable partial hyper-link and when determining membership in the partial hyper-link and clause sets. We say P is the *smallest instance first hyper-link proof procedure* defined by $\geq$. We refer to $\geq$ as the *hyper-link ordering* of P. ∎

The smallest instance first hyper-link proof procedures are sound and logically complete.

Theorem 4.1 (Soundness of Smallest Instance First Hyper-Linking) Every smallest instance first hyper-link proof procedure is sound.

Proof: Let P be an arbitrary smallest instance first hyper-link proof procedure and F an arbitrary closed formula such that $\vdash_P F$. Let $S_0, \ldots, S_n$ be the sequence of sets of clauses generated by running P on $(\{\}, F)$. We note that $S_0, \ldots, S_n$ is a hyper-link sequence and that S_n is propositionally unsatisfiable. By Theorem 2.45, S_n is unsatisfiable. Thus, by the soundness of hyper-linking (Theorem 2.46), S_0 is unsatisfiable. We note that S_0 is the Skolemization of $\sim F$. So, by Theorem 2.7, $\sim F$ is unsatisfiable. Thus, by Theorem 2.3, $\models F$. ∎

Definition 4.3 Let S be a set of clauses. We say S is *hyper-linkable* if there exists an instance C of a hyper-link in S such that S does not contain a variant of C. ∎

Lemma 4.2 Let P be a smallest instance first hyper-link proof procedure defined by the hyper-link ordering $\geq$ on partial hyper-links, P' the algorithm defined by removing the propositional unsatisfiability test from P, S a finite set of closed formulas, F a closed formula, and $S_0, S_1, \ldots$ the sequence of sets of clauses generated by running P' on (S, F). Then for $i \geq 0$, S_{i+1} exists iff S_i is hyper-linkable.

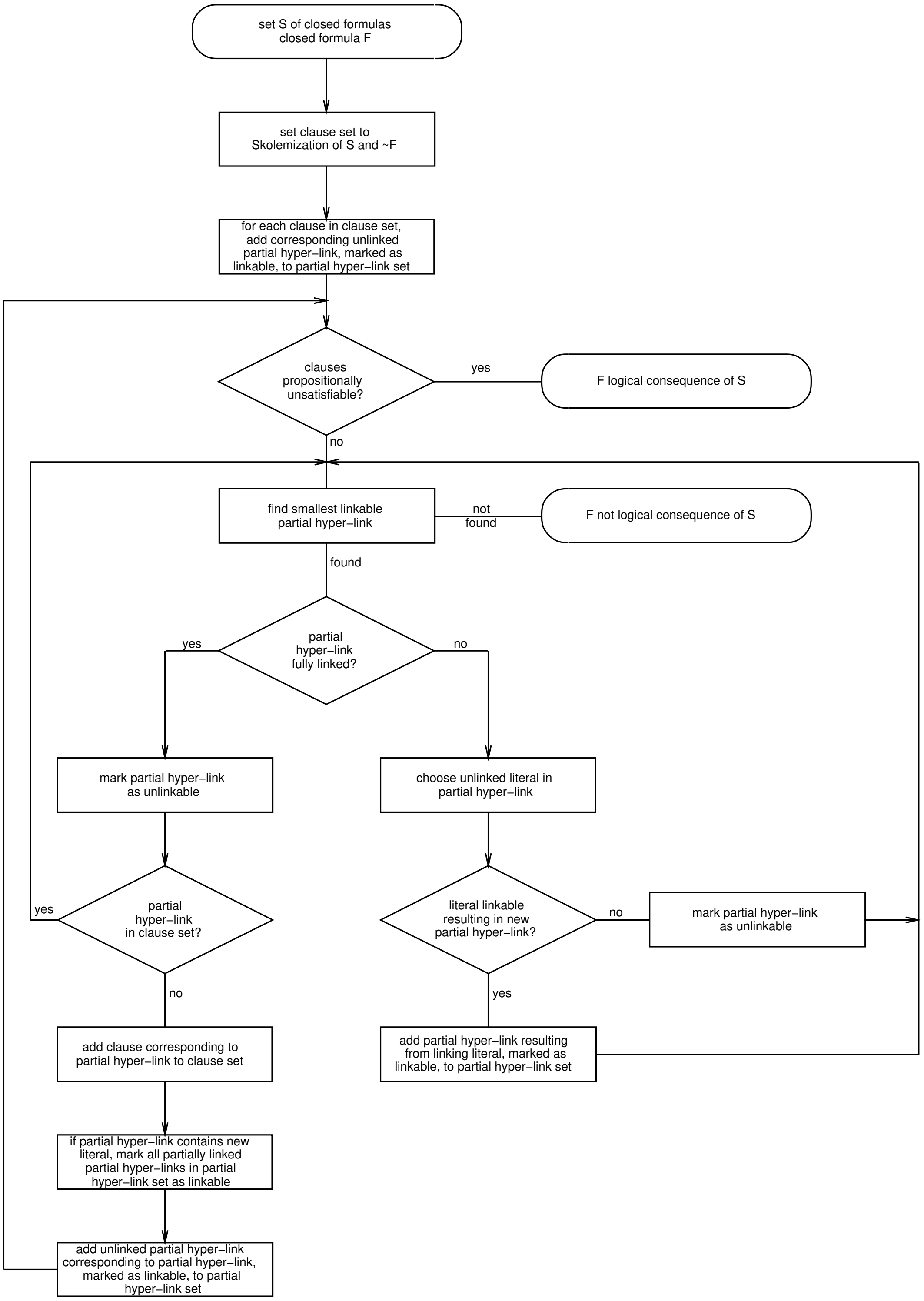

Figure 4.1: Smallest Instance First Hyper-Linking Proof Procedures

Proof: Let S_{i+1} be an arbitrary set of clauses in $S_1, S_2, \ldots$. We note that S_{i+1} is $S_i \cup \{C\}$ where C is an instance of a hyper-link in S_i such that S_i does not contain a variant of C. Thus, S_i is hyper-linkable.

Let S_i be an arbitrary hyper-linkable set of clauses in $S_0, S_1, \ldots$. We note that S_0 is the Skolemization of $S \cup {\sim}F$. So, by Theorem 2.7, S_0 is finite. By induction, S_j is finite for all $j \geq 0$. So, S_i is finite. Since S_i is finite, there exist only a finite number of links in S_i. So, since S_i is hyper-linkable, P' must eventually generate S_{i+1}. ∎

Lemma 4.3 Let P be a smallest instance first hyper-link proof procedure defined by the hyper-link ordering $\geq$ on partial hyper-links, P' the algorithm defined by removing the propositional unsatisfiability test from P, S a finite set of closed formulas, F a closed formula, and $S_0, S_1, \ldots$ the sequence of sets of clauses generated by running P' on (S, F). Then for $i \geq 0$, S_{i+1}, if it exists, is $S_i \cup \{C\}$ where C is a $\geq$-smallest instance of a hyper-link in S_i.

Proof: Let S_{i+1} be an arbitrary set of clauses in $S_1, S_2, \ldots$. We note that S_{i+1} is $S_i \cup \{C\}$ where C is the instance of a hyper-link in S_i corresponding to a fully linked $\geq$-smallest partial hyper-link. By the definition of $\geq$, no partial hyper-link $\geq$-smaller than C can be generated by further linking of partial hyper-links. Thus, C is a $\geq$-smallest instance of a hyper-link in S_i. ∎

Lemma 4.4 Let P be a smallest instance first hyper-link proof procedure defined by the hyper-link ordering $\geq$ on partial hyper-links and P' the algorithm defined by removing the propositional unsatisfiability test from P. Then every sequence $S_0, S_1, \ldots$ of sets of clauses generated by P' is a fair hyper-link sequence.

Proof: Let $>$ be the strict part of $\geq$, $S_0, S_1, \ldots$ an arbitrary sequence of sets of clauses generated by P', S_i an arbitrary set of clauses in $S_0, S_1, \ldots$, and C an arbitrary instance of a hyper-link in S_i. By induction, C is an instance of hyper-link in S_j for all $j \geq i$. Since $>$ is well-founded, there exist only a finite number of instances of hyper-links $>$-smaller than C. So, by Lemmas 4.2 and 4.3, P' must eventually generate a set S_j of clauses containing a variant of C. Thus, since C is an arbitrary instance of a hyper-link of an arbitrary set S_i of clauses, $S_0, S_1, \ldots$ is a fair hyper-link sequence. ∎

Theorem 4.5 (Completeness of Smallest Instance First Hyper-Linking) Every smallest instance first hyper-link proof procedure is logically complete.

Proof: Let P be an arbitrary smallest instance first hyper-link proof procedure, P' the algorithm defined by removing the propositional unsatisfiability test from P, F an arbitrary valid closed formula, and $S_0, S_1, \ldots$ the sequence of sets of clauses generated by running P' on $(\{\,\}, F)$. We note that S_0 is the Skolemization of $\sim F$. So, by Theorems 2.3 and 2.7, S_0 is unsatisfiable. By Lemma 4.4, $S_0, S_1, \ldots$ is a fair hyper-link sequence. Thus, by the completeness of hyper-linking (Theorem 2.47), there exists an S_n in $S_0, S_1, \ldots,$ such that S_n is propositionally unsatisfiable. Therefore, running P on $(\{\,\}, F)$ generates the sequence $S_0, \ldots, S_n$ of sets of clauses, after which P terminates with an indication of "F is a logical consequence of S". Thus, $\vdash_P F$. ∎

4.2 Choosing Which Literal to Link

Smallest instance first hyper-link proof procedures, in addition to having a choice in the hyper-link ordering used to order partial hyper-links, also have a choice in the order in which literals from a nucleus are linked. We note, however, that the application of the hyper-link operation does not depend on the order in which the literals are linked. Thus, allowing the smallest instance first hyper-link proof procedures to arbitrarily choose at link time which literal to link results in an exponential amount of duplicate work. This can be seen by considering the hyper-link instance generated by hyper-linking the nucleus $\{\, L_1, \ldots, L_n \,\}$ with electrons $M_1, \ldots, M_n$. There are $n!$ permutations in which the individual links can be computed, all resulting in the same hyper-link instance. We eliminate this duplicate work by choosing which literal to link when the partial hyper-link is added to the partial hyper-link set, rather than when linking. In this way, a smallest instance first hyper-link proof procedure performs only one of the $n!$ permutations in generating each hyper-link instance.

The number of attempted links (unifications) performed when hyper-linking a nucleus can vary greatly depending on the order in which the literals are linked. For example, consider the nucleus $\{\, L_1, L_2, L_3 \,\}$. Say that there are 1,000 electrons and that L_1 has one link, L_2 has 10 links, and L_3 has 100 links. For simplicity, say that the links of each literal are independent of each other. This means that there are 1,000 hyper-link instances. Linking the literals in the order L_1 L_2 L_3 requires 12,000 unifications as follows. Linking L_1 requires 1,000 unifications, of which one succeeds. Linking L_2 requires $1 \times 1,000 = 1,000$ unifications, of which 10 succeed. Linking L_3 requires $10 \times 1,000 = 10,000$ unifications, of which 1,000 succeed. On the other hand, linking the literals in the order L_3 L_2 L_1 requires 1,101,000 unifications as follows. Linking L_3 requires 1,000 unifications, of which 100 succeed. Linking L_2 requires $100 \times 1,000 = 100,000$ unifications, of which 1,000 succeed. Linking L_1 requires $1,000 \times 1,000 = 1,000,000$ unifications, of which 1,000 succeed. We note that linking literals in order of fewest links results in the smallest number of unifications.

Note that in general, links from different literals are not independent of each other, as we assumed above. This complicates choosing the best order in which to link. For example, consider the nucleus $\{\, p(x), q(y), r(x) \,\}$ and electrons $\sim p(a)$, $\sim q(b)$, $\sim q(c)$, $\sim r(d)$, $\sim r(e)$, and $\sim r(f)$. Note that $p(x)$ has one link, $q(y)$ has two links, and $r(x)$ has three links. Linking in the order $p(x)$ $q(y)$ $r(x)$ results in 24 unifications. However, linking in the order $p(x)$ $r(x)$ $q(y)$ results in only 12 unifications. The reason for this can be seen by considering the partial hyper-link after linking $p(x)$, which is $\{\, p(a), q(y), r(a) \,\}$. Note that while $r(x)$ has three links, $r(a)$ has no links. Thus, instead of linking $q(y)$ next, we should link $r(a)$. We note that in order to minimize the number of unifications, it is necessary to dynamically choose which unlinked literal to link next after each link is performed. We do this by choosing which literal to link next when adding partial hyper-links to the partial-link set.

We note that linking a ground literal does not modify any of the literals in the partial hyper-link. Thus, when linking a ground literal, we need not compute all links. Instead, we need only determine if the ground literal is linkable.

We use an overscore to indicate that a literal in a partial hyper-link is the linkable literals. For example, $\{\, \overline{p(a)}, q(b), \underline{r(x)} \,\}$ is a partially linked partial hyper-link with linkable literal $p(a)$ and linked literal $r(x)$.

The preliminary implementation of smallest instance first hyper-linking does choose which literal to link when adding a partial hyper-link to the partial hyper-link set. For efficiency, it does not count the number of possible links when choosing which literal to link; rather, it chooses the smallest unlinked literal since smallest literals should, in general, have fewer links than larger literals. The preliminary implementation does include the optimization for ground literals. Also when choosing the which literal to link, the preliminary implementation chooses ground literals over non-ground literals.

4.3 Support Strategies

Backward and forward support performed quite well in round-by-round hyper-linking and should prove just as useful for smallest instance fist hyper-linking.

We add backward support to smallest instance first hyper-linking by only performing backward supported links. Similarly, we add forward support to smallest instance first hyper-linking by only performing forward supported links. We do not add user support to smallest instance first hyper-linking for two reasons: (1) it requires human intervention in that the user must mark selected input clauses as user supported and (2) user support is less restrictive than backward and forward support.

The preliminary implementation of smallest instance first hyper-linking includes backward and forward support.

4.4 Tautology Deletion

The overhead of testing for tautology deletion is quite small and the savings when tautologies are deleted can be quite large. Thus, tautology deletion should be of benefit to smallest instance first hyper-linking.

We add tautology deletion to smallest instance hyper-linking by ensuring that a partial hyper-link is not a tautology before adding it to the partial hyper-link set. Note that testing for tautologies in this way encompasses both tautology deletion on clauses and early tautology deletion on partial hyper-links.

The preliminary implementation of smallest instance first hyper-linking includes tautology deletion.

4.5 Unit Subsumption

Unit subsumption worked quite well in round-by-round hyper-linking and should work as equally well in smallest instance first hyper-linking.

We add unit subsumption to smallest instance first hyper-linking in two ways. We first ensure that a partial hyper-link is not subsumed by a unit clause before adding it to the partial hyper-link set. We refer to this test as *forward unit subsumption*. On consequence of forward unit subsumption is that it is unnecessary to perform hyper-linking on unit clauses, as every hyper-link instance of a unit clauses is subsequently unit subsumed by its nucleus. To address the problem which occurs when unit clauses are unit subsumed by their nuclei, we test the clause set for a unit conflict after adding a unit clause.

Definition 4.4 Let $(\{L\}, \{M\})$ be a pair of unit clauses. We say $(\{L\}, \{M\})$ is a *unit conflict* if L and M link. ■

Theorem 4.6 Let S be a set of clauses and C and D clauses in S. If (C, D) is a unit conflict, then S is unsatisfiable. ■

When adding a unit clause to the clause set, we delete all clauses or partial hyper-links which are subsumed by the unit clause. We refer to this test as *backward unit subsumption*. Note that when deleting a clause from the clause set, we also delete any partial hyper-link which depends on the deleted clause, either as a nucleus or as a source of an electron. The deletion of dependent partial-hyper links when deleting a clause is fairly complicated in that a partial hyper-link may result from the linking of more than more nucleus. Also, an electron of a partial hyper-link may come from more than one clause. Thus, we only delete a partial hyper-link when its last nucleus is deleted or when one of its electrons is deleted. We only delete an electron when the last clause containing the electron is deleted. When using backward, forward, or user supported hyper-linking, we treat an electron occurring in both supported and an unsupported clauses as two electrons, one as a supported electron and one as an unsupported electron.

Note that forward and backward unit subsumption encompass both unit subsumption of clauses and early unit subsumption of partial hyper-links.

The preliminary implementation of smallest instance first hyper-linking includes a simplified form of unit subsumption which only performs unit subsumption on clauses. The preliminary implementation does include the test for unit conflicts.

4.6 Unit Literal Deletion

Unit literal deletion worked quite well in round-by-round hyper-linking and should work as equally well in smallest instance first hyper-linking. In fact, results obtained from the preliminary implementation of smallest instance first hyper-linking indicate that unit literal deletion is more important for smallest instance first hyper-linking than we initially realized.

We add unit literal deletion to smallest instance first hyper-linking in two ways. When adding a partial hyper-link to the partial hyper-link set, we delete any literals which are contradictory instances of a unit clause. We refer to deletion of literals in this way as *forward unit literal deletion*. Note that with the addition of forward literal deletion, we are no longer assured that instance generated via linking is larger than the original partial hyper-link, since the instance may have one or more literals deleted. We address this by performing all unit links as each partial hyper-link is added to the partial hyper-link set.

When adding a unit clause to the clause set, we delete literals from clauses which are contradictory instances of the unit clause. Note that when deleting literals from a clause, we delete any partial hyper-links dependent on the original clause, as described earlier. We also add an unlinked partial hyper-link corresponding to the new clause, marked as linkable, to the partial hyper-link set. In addition to deleting literals from clauses, we also delete linked literals from partial hyper-links which are contradictory instances to the unit clause. We refer to deletion of literals from clauses and partial hyper-links in this way as *backward unit literal deletion*.

Note that forward and backward unit literal deletion encompass both unit literal deletion of clauses and embedded unit literal deletion of partial hyper-links. In fact, forward and backward unit literal deletion are stronger than embedded unit literal deletion in that they can delete linked literals from partial hyper-links which are linked with non-unit clauses.

The preliminary implementation of smallest instance first hyper-linking includes a simplified form of unit literal deletion which only performs unit literal deletion on clauses. The preliminary implementation of smallest instance first hyper-linking does not perform unit links as each partial hyper-link is added to the partial hyper-link set.

4.7 UR Resolution

UR resolution, although not logically complete in general, performed quite well on selected theorems in round-by-round hyper-linking. It should perform as equally well in smallest instance first hyper-linking.

We can modify smallest instance first hyper-linking to perform UR resolution rather than hyper-linking. When adding an unlinked partial hyper-link corresponding to an n-literal clause to the partial hyper-link set, instead of adding a single partial hyper-link, we add n partial hyper-links corresponding to the n-literal clause, in which each partial hyper-link has a different literal marked as linked. For example, adding the unlinking partial hyper-link $\{\, p(a), q(b), r(x) \,\}$ to the partial hyper-link set, we would actually added the following three partial hyper-links: $\{\, \underline{p(a)}, \overline{q(b)}, r(x) \,\}$, $\{\, \overline{p(a)}, \underline{q(b)}, r(x) \,\}$, and $\{\, \overline{p(a)}, q(b), \underline{r(x)} \,\}$. When linking partial hyper-links, we only link with unit clauses.

The preliminary implementation of smallest instance first hyper-linking includes support for UR resolution.

4.8 Ground Unit Generation

We include a discussion on ground unit generation since it was included in round-by-round hyper-linking.

We add ground unit generation to smallest instance first hyper-linking by applying ground unit generation when adding a partial hyper-link to the partial hyper-link set. Note that our use of ground unit generation is more powerful than Lee's in that we apply ground unit generation to partial hyper-links as well as clauses (fully linked partial hyper-links).

The preliminary implementation of smallest instance first hyper-linking does not include ground unit generation.

4.9 Proper Subsumption

Proper subsumption is an extension to unit subsumption. We include a description of proper subsumption as it is of theoretical interest; it most likely will not be practical due to the exponential cost in performing the test.

We add proper subsumption to smallest instance first hyper-linking by extending unit subsumption. In this way, forward unit subsumption becomes *forward proper subsumption*, and backward unit subsumption becomes *backward proper subsumption*. We note that proper subsumption may not be practical for clauses having many literals due to the combinatorial number of subsets required to be examined. For example, testing if a 10-literal clause C is properly subsumed by a 5-literal clause D requires, in the worse case, that $\binom{10}{5} = 252$ subsets of C be examined. Furthermore, determining if one 5-literal clause is an instance of another 5-literal clause requires, in the worse case, that $5! = 120$ permutations be examined[16]. Thus, $252 \times 120 = 30,240$ instance tests are required, in the worse case, to determine if a 10-literal clause is properly subsumed by a 5-literal clause.

The preliminary implementation of smallest instance first hyper-linking does not include proper subsumption.

4.10 Small Proof Checking

Small proof checking was developed by Lee and Plaisted to reduce the number of rounds required in round-by-round hyper-linking to find a proof. The basic idea of small proof checking is to periodically test for applications of the hyper-link operation involving only unit links. The implementation of unit literal deletion given in Section 4.6 performs all unit links as each partial hyper-link is added to the partial hyper-link set, thus simulating simple small proof checking. It is unnecessary for smallest instance first hyper-linking to explicitly check for small proofs.

[16]Note that by appropriately ordering literals within the clauses, it might be possible to reduce the number of permutations tested when determining if one clause is an instance of another.

4.11 Predicate Replacement

Predicate replacement allowed round-by-round hyper-linking to prove many difficult theorems, in particular difficult theorems from von Neumann-Bernays-Gödel set theory, which it would have otherwise been unable to prove. Predicate replacement should work as equally well for smallest instance first hyper-linking.

We add predicate replacement to smallest instance first hyper-linking by treating replacement rules as partial hyper-links in which the non-distinguished literals are marked as linked. We also mark electrons as to whether they occur only as instances of distinguished literals. When linking partial hyper-links corresponding to replacement rules, we do not use electrons occurring only as instances of distinguished literals. We also ensure that the substitution of the link maps each variable in the electron to either a variable or a constant. Note that since smallest instance first hyper-linking generates smallest instances first, it is unnecessary that the set of replacement rules be terminating.

The preliminary implementation of smallest instance first hyper-linking does not include predicate replacement.

4.12 Sliding Priority

Sliding priority was developed by Lee and Plaisted to dynamically control the size of the search space based on the amount work performed. Smallest instance first hyper-linking inherently controls the size of search space by generating smallest instances first; no explicit mechanism for controlling the size of the search space in required. Therefore, we feel that it unnecessary to add sliding priority (or any other priority control method) to smallest instance first hyper-linking.

4.13 Automated Clause Splitting

Hyper-linking does not work well on clauses having many literals. Clause splitting provides an attractive method for eliminating long clauses and should work well with smallest instance first hyper-linking.

We add automated clause splitting to smallest instance first hyper-linking by splitting long clauses as they are added to the clause set. Note that only input clauses need to be split as neither hyper-linking nor any of the enhancements increase the number of literals in a clause.

The preliminary implementation of smallest instance first hyper-linking does not include automated clause splitting.

4.14 Brand's E-Modification

Brand's E-modification is not well suited for round-by-round hyper-linking as the number of hyper-link instances of a clause is exponential in the number of literals in the clause. However, smallest instance first hyper-linking controls the generation of hyper-link instances by performing links one at a time. Smallest instance first generation of hyper-link instances should alleviate the problem with the exponential number of hyper-link instances as it is no longer necessary to generate all hyper-link instances of a clause.

We add Brand's E-modification to smallest instance first hyper-linking by applying the E-modification to the results of Skolemization and by adding $\{ x = x \}$ and $\{ x \neq y, x \neq z, y = z \}$ to the transformed clause set before the creation of the partial hyper-link set.

The preliminary implementation of smallest instance first hyper-linking does not include Brand's E-modification.

4.15 Automated Group Detection

Automated group detection worked quite well when adding equality support to round-by-round hyper-linking. Automated group detection should as equally well in smallest instance first hyper-linking.

The addition of group detection to smallest instance first hyper-linking is quite straightforward. We simply search for a group whenever an equation is added to the clause set. As in extended CLIN, we look for either the associativity, left inverse, and left identity axioms or associativity, right inverse, and right identity axioms. We also allow the binary operation, inverse operation, and identity element to be arbitrary terms containing two, one, and zero distinct variables. When a group is found, we add the appropriate equations and replacement rules. Note that to obtain full advantage of group detection, both predicate replacement and unfailing completion equality support (see Chapter 5) must have been added to smallest instance first hyper-linking.

The preliminary implementation of smallest instance first hyper-linking does not include automated group detection.

Chapter 5

Unfailing Completion Equality Support

In Chapter 3 we described a number of inefficiencies encountered when adding term rewriting to round-by-round hyper-linking. Furthermore, term rewriting may need to be restricted when applying Brand's E-modification to the input clause set in order to maintain completeness of round-by-round hyper-linking. We present an alternative method, which we refer to as *unfailing completion equality support*, for adding equality support to smallest instance first hyper-linking. Unfailing completion equality support is based on the use of unfailing completion and the following inference rule, where E is a set of equations and $>$ is a termination ordering (recall that $C\theta[l\theta \to r\theta]_p$ denotes the clause obtained by replacing the term $l\theta$ at position p by the term $r\theta$).

$E_>$**-Narrowing:** $\dfrac{C[t]_p \qquad \{\, l \doteq r \,\}}{C\theta[l\theta \to r\theta]_p}$ where t is a non-variable subterm of C, $\{\, l \doteq r \,\}$ is an equation in E, θ is a most general unifier of t and l, and $l\theta > r\theta$

We discuss unfailing completion equality support in the context of smallest instance first hyper-linking. However, unfailing completion equality support is completely general and can be added to other clause-based refutation proof procedures in a manner similar to that presented in this chapter.

Smallest instance first hyper-linking with unfailing completion equality support attempts to show that a closed formula F is an E-logical consequence of a finite set S of closed formulas. Note that there are many smallest instance first hyper-linking proof procedures with unfailing completion equality support, each differing in the choice of the termination ordering $>$ and partial ordering $\rhd$ on terms used in unfailing completion and in the hyper-link ordering $\succeq$ used in choosing the smallest equation, clause, or partial hyper-link. Smallest instance first hyper-link proof procedures with unfailing completion equality support also differ in the strategy used in applying the unfailing completion inference rules when reducing sets of equations.

Smallest instance first hyper-linking with unfailing completion equality support consists of three components: unfailing completion, narrowing, and hyper-linking. Unfailing completion generates a ground canonical term rewriting system which rewrites the unsatisfiable finite subset of ground instances of Herbrand's Theorem into an unsatisfiable set of clauses; narrowing is used to fixed up the links which are broken by term rewriting, thus allowing term rewriting to be applied to non-ground clauses as well as ground clauses; and smallest instance first hyper-linking repeatedly generates instances of clauses in an attempt to find the unsatisfiable finite subset of ground instances of Herbrand's Theorem.

In the following description of smallest instance first hyper-linking with unfailing completion equality support, we use S_s to denote the Skolemization of S and $\sim F$ and S_d to denote the set equations derived by hyper-linking with unit literal deletion. The unfailing completion component uses two sets E and E' of equations; the narrowing component uses two sets N and N' of clauses; and the hyper-linking component uses a set S' of clauses and a set P of partial hyper-links.

The goal of smallest instance first hyper-linking with unfailing completion equality support is to fairly apply unfailing completion, narrowing, and hyper-linking so that in the possibly infinite limit, $E'_>$ becomes a ground canonical ordered rewriting system equivalent to S_d, N' becomes the $E'_>$-narrowing closure of E-mod$(S_s) \cup \{ \{ x = x \}, \{ x \neq y, x \neq z, y = z \} \}$, and S' becomes the hyper-link closure of the $E'_>$-irreducible clauses of N'. During the execution of smallest instance first hyper-linking with unfailing completion equality support, E is used to hold the equations of S_d as well as the equations corresponding to extended critical pairs of E', N is used to hold the clauses in E-mod$(S_s) \cup \{\{x = x\}, \{x \neq y, x \neq z, y = z\}\}$ as well the clauses derived from the $E'_{>}$-narrowing of N', and P is used to hold $E'_{>}$-irreducible partial hyper-links in a manner similar to smallest instance first hyper-linking.

Initially, N is set to E-mod$(S_s) \cup \{\{x = x\}, \{x \neq y, x \neq z, y = z\}\}$ and E, E', N', S', and P are set to the empty set. Smallest instance first hyper-linking with unfailing completion equality support repeatedly chooses the $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P. If no such equation, clause, or partial hyper-link exists, smallest instance first hyper-linking with unfailing completion equality support stops and reports that F is not an E-logical consequence of S. If the smallest equation, clause, or partial hyper-link chosen is an equation from E, the equation is marked as processed and added to E', E' is reduced via unfailing completion[17], and E, N, N', S', and P are updated to reflect the changes in E'. If the smallest equation, clause, or partial hyper-link chosen is a clause from N, the clause is marked as processed and added to N' and N, S', and P are updated to reflect the change in N'. If the smallest equation, clause, or partial hyper-link chosen is a fully linked partial hyper-link corresponding to an equation[18], the partial hyper-link is marked as unlinkable and the corresponding equation added to E. If the smallest equation, clause, or partial hyper-link chosen is a fully linked partial hyper-link corresponding to a non-equation, the partial hyper-link is marked as unlinkable, the corresponding non-equation is added to S', and P is updated to reflect the change to S'. If the smallest equation, clause, or partial hyper-link chosen is a partially linked partial hyper-link, the partial hyper-link is linked[19] and the resulting partial hyper-link is added to P. Each time the set S' is modified, a propositional unsatisfiability test is performed on S'. If S' is determined to be propositionally unsatisfiable, smallest instance first hyper-linking with unfailing completion equality support stops and reports that F is an E-logical consequence of S.

[17] Instead of maintaining unordered equations and rewrite rules in separate sets, both are maintained in E' as equations with rewrite rules specially marked.

[18] A partial hyper-link may correspond to an equation due to unit literal deletion.

[19] The method for choosing which literal to link described in Section 4.2 is applied here.

Smallest instance first hyper-linking is guaranteed to find a proof, if one exists, whenever $>$ has the full invariance property. We present a general sketch of why, with a formal proof of both soundness and logical completeness to follow later in the chapter. Assume that S_s is E-unsatisfiable. Since, in the limit, $E'_>$ is ground canonical and N' is closed under $E'_>$-narrowing, the set of $E'_>$-irreducible clauses in N' is unsatisfiable. Thus, since $>$ has the full invariance property and since hyper-linking is logically complete, S' eventually becomes propositionally unsatisfiable.

We now present a more detailed description of smallest instance first hyper-linking with unfailing completion equality support.

1. Set N to E-mod$(S_s) \cup \{\{x = x\}, \{x \neq y, x \neq z, y = z\}\}$ where S_s is the Skolemization of S and $\sim F$.
2. Set E, E', N', S' and P to the empty set.
3. Set D to the $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P. If no such equation, clause, or partial hyper-link exists, stop and report that F is not an E-logical consequence of S.
4. If D is an equation from E, do the following:
   a) Mark D as processed.
   b) Set D' to an $E'_>$-normal form of D.
   c) If D' does not have the form $\{s = s\}$ and is not in E', do the following:
      i) Add D' to E'.
      ii) Reduce the equations in E' by repeated applications of the unfailing completion inference rules delete, compose, compose$_2$, simplify, simply$_2$, orient, collapse, and collapse$_2$.
      iii) Remove all equations in E derived solely as extended critical pairs of equations no longer in E' as a result of step ii.
      iv) Add to E the equations, marked as unprocessed, corresponding to all extended critical pairs of E' coming from the equations added to E' as a result of step ii.
      v) Remove all clauses in N and N' derived, directly or indirectly, from the equations no longer in E' as a result of step ii.
      vi) Add to N the results, marked as unprocessed, of all single applications of narrowing to the clauses remaining in N' as a result of step v with the equations added to E' as a result of step ii.
      vii) Remove all $E'_>$-reducible clauses in S' and $E'_>$-reducible partial hyper-links in P and all clauses in S' and all partial hyper-links in P derived, directly or indirectly, from $E'_>$-reducible clauses in S' and $E'_>$-reducible partial hyper-links in P.

viii) Remove all equations from E derived solely from clauses and and partial hyper-links no longer in S' and P, respectively, as a result of step vii.

d) Go to step 3.

5. If D is a clause from N, do the following:
   a) Mark D as processed.
   b) Add D to N'.
   c) Add to N the results of all single applications of narrowing of D with the equations in E', marked as unprocessed.
   d) Set D' to the unit deletion of D in S'.
   e) If D' is not in S' and is $E'_{>}$-irreducible, do the following:
      i) If D' contains a literal not in some clause of S', mark all partially linked partial hyper-links in P as linkable.
      ii) Add D' to S'.
      iii) Determine if S' is propositionally unsatisfiable. If so, stop and report that F is an E-logical consequence of S.
      iv) Add the unlinked partial hyper-link corresponding to D', marked as linkable, to P.
   f) Go to step 3.
6. If D is a fully linked partial hyper-link from P corresponding to an equation which is not an instance of $\{ x = x \}$, do the following:
   a) Mark D as unlinkable.
   b) Add the equation, marked as unprocessed, corresponding to D to E.
   c) Go to step 3.
7. If D is a fully linked partial hyper-link from P corresponding to a non-equation or an instance of $\{ x = x \}$, do the following:
   a) Mark D as unlinkable.
   b) If D is not in S', do the following:
      i) If D contains a literal not in some clause of S', mark all partially linked partial hyper-links in P as linkable.
      ii) If D is a unit clause, apply the unit literal deletion by D to S'.
      iii) Add D to S'.
      iv) Determine if S' is propositionally unsatisfiable. If so, stop and report that F is an E-logical consequence of S.
      v) Add the unlinked partial hyper-link corresponding to D, marked as linkable, to P.
   c) Go to step 3.
8. If D is a partially linked partial hyper-link from P, do the following:
   a) Choose an unlinked literal L in D and literal M in a clause in S' such that L and M link with most general unifier θ, the unit literal deletion of $D\theta$ is not in P, $D\theta$ is not unit subsumed, and $D\theta$ is $E'_{>}$-irreducible. If no such literals L and M exist, do the following:
      i) Mark D as unlinkable.
      ii) Go to step 3.

b) Add the unit literal deletion of $D\theta$, marked as linkable, to P.
c) Go to step 3.

This process is illustrated in Figure 5.1. Figure 5.2 presents the detailed processing which occurs when the $\succeq$-smallest equation, clause, or partial hyper-link is an equation from E. Figure 5.3 presents the detailed processing which occurs when the $\succeq$-smallest equation, clause, or partial hyper-link is a clause from N. Figure 5.4 presents the detailed processing which occurs when the $\succeq$-smallest equation, clause, or partial hyper-link is a fully linked partial hyper-link from P corresponding to an equation which is not an instance of $\{ x = x \}$. Figure 5.5 presents the detailed processing which occurs when the $\succeq$-smallest equation, clause, or partial hyper-link is a fully linked partial hyper-link from P corresponding to a non-equation or an instance of $\{ x = x \}$. Figure 5.6 presents the detailed processing which occurs when the $\succeq$-smallest equation, clause, or partial hyper-link is a partially linked partial hyper-link from P.

Definition 5.1 Let P' be an algorithm, possibly non-terminating, implementing the process presented in Figures 5.1–5.6 using the termination ordering $>$ and partial ordering $\rhd$ on terms when applying the unfailing completion inference rules and the hyper-link ordering $\succeq$ when choosing the $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P and when determining membership in E, E', N, N', S' and P. We say P' is a *smallest instance first hyper-link proof procedure with unfailing completion equality support* defined by $>$, $\rhd$, and $\succeq$. We refer to $>$ as the *termination ordering* of P', $\rhd$ as the *specialization ordering* of P', and $\succeq$ as the *hyper-link ordering* of P'. ∎

The smallest instance first hyper-link proof procedures with unfailing completion equality support are sound. Also, smallest instance first hyper-link proof procedures with unfailing completion equality support whose termination orderings have the full invariance property are logically complete.

Definition 5.2 Let $>$ be a termination ordering, E and E' sets of equations, N and N' sets of non-equations plus possibly $\{ x = x \}$, S' a set of $E'_>$-irreducible clauses, and P a set of $E'_>$-irreducible partial hyper-links. We say the tuple (E, E', N, N', S', P) is an *unfailing completion equality support tuple*. ∎

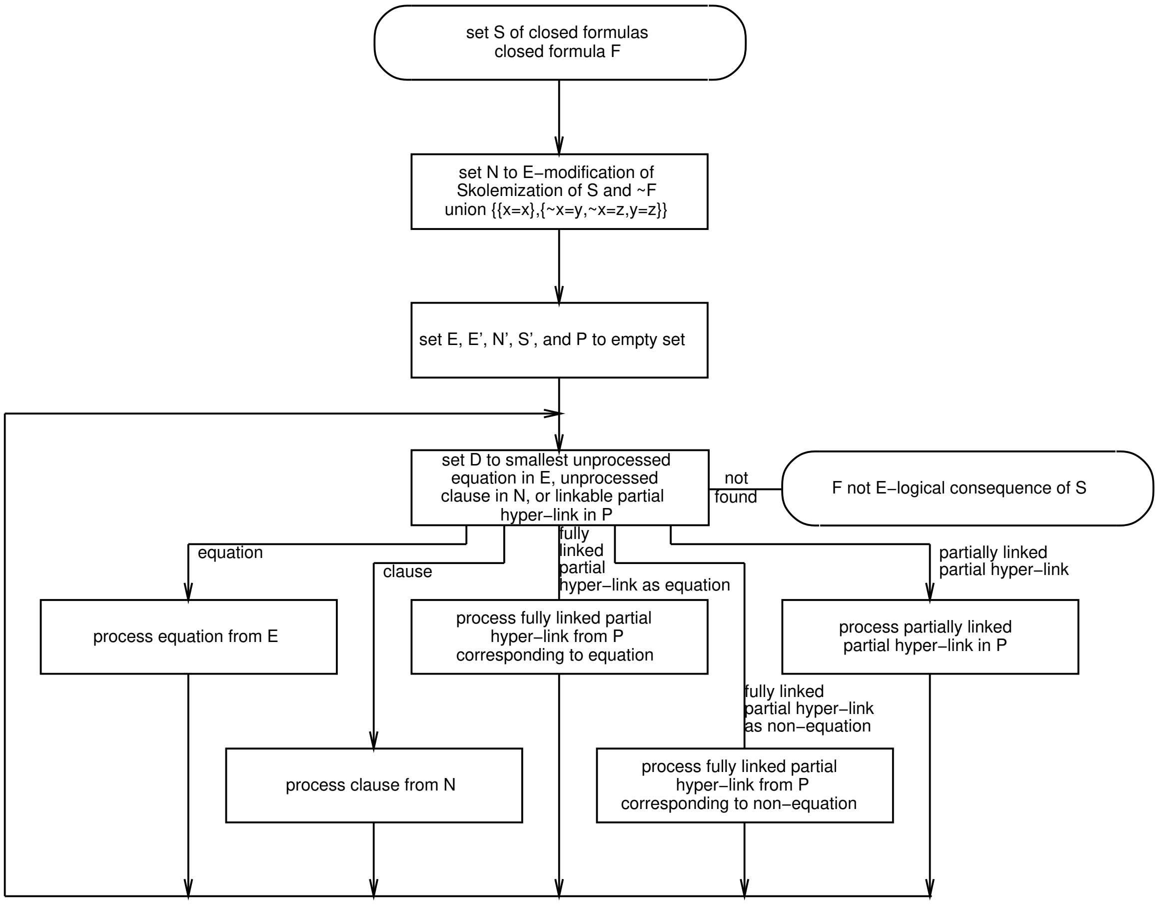

Figure 5.1: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support

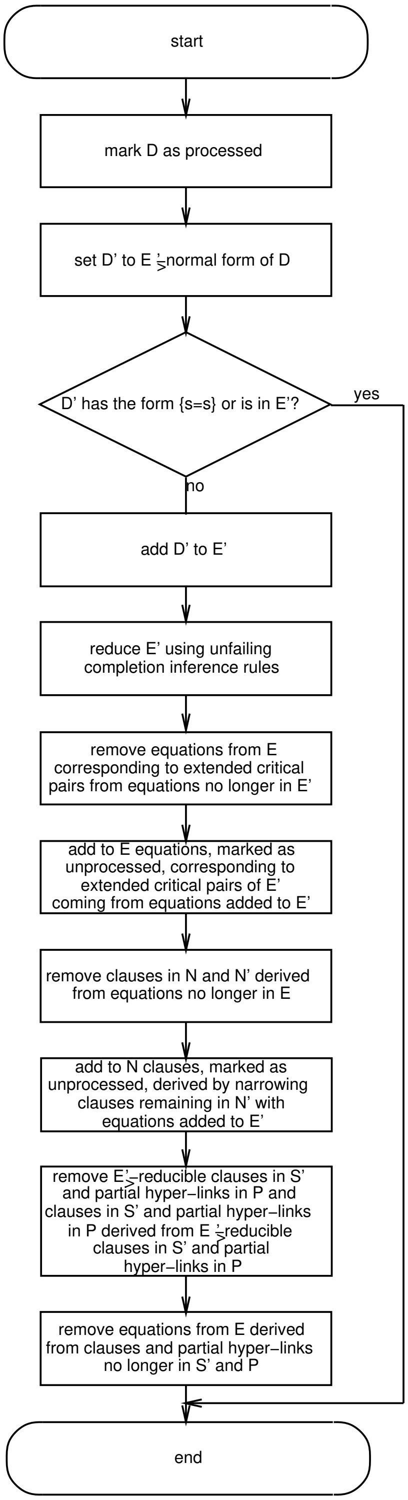

Figure 5.2: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support — Processing Equation from E

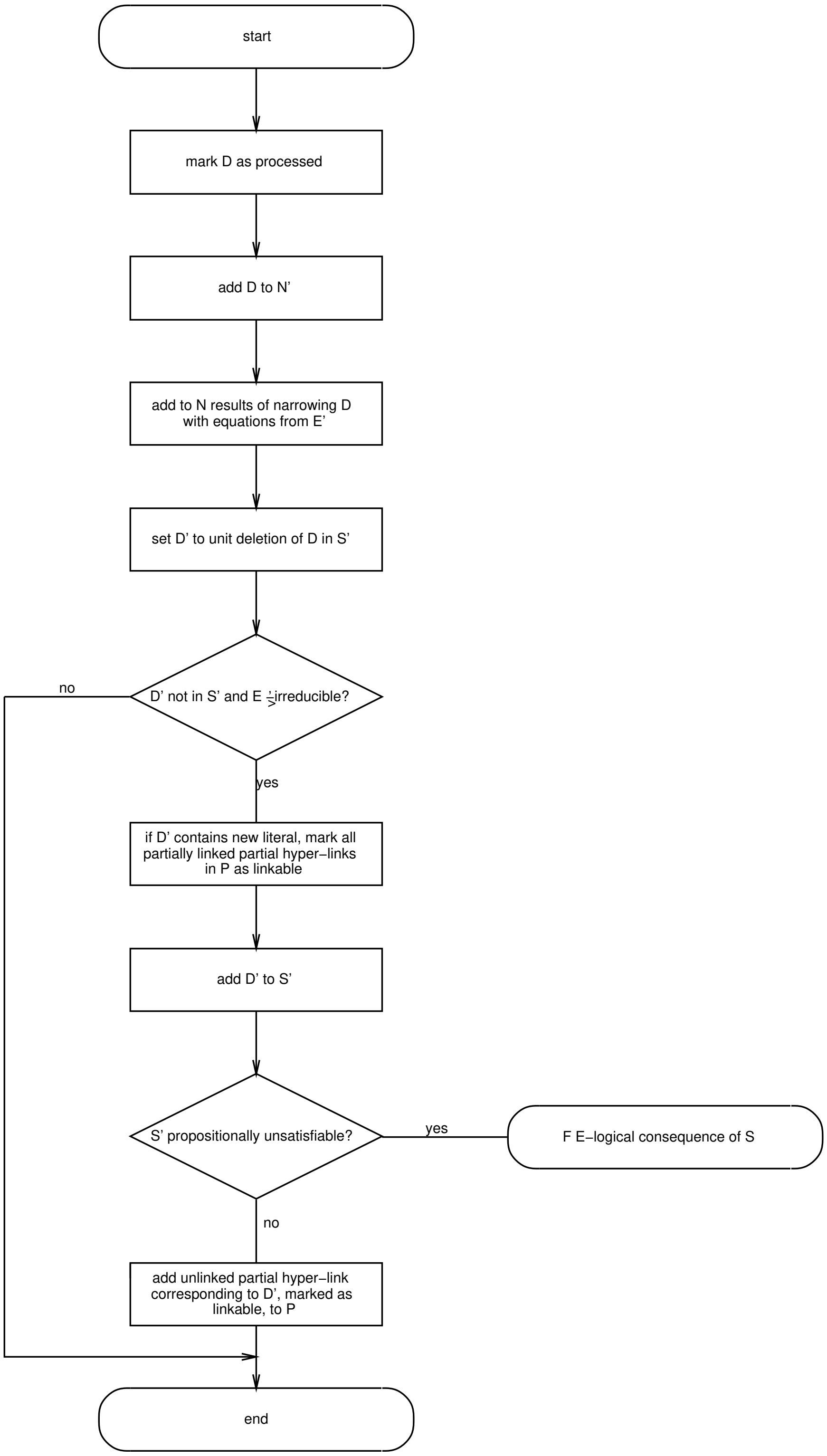

Figure 5.3: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support — Processing Clause from N

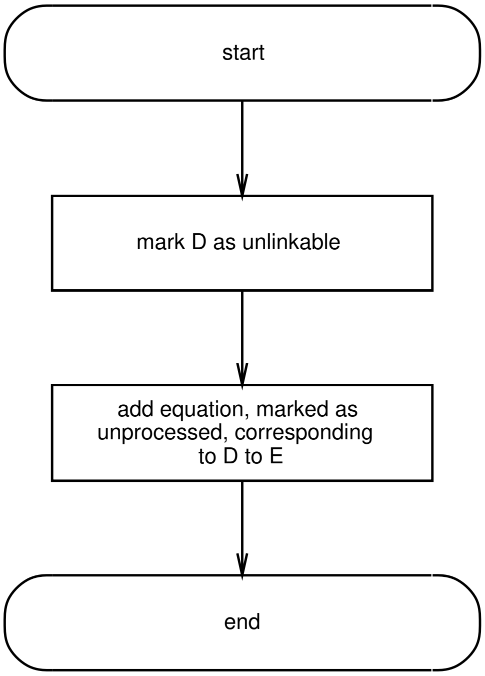

Figure 5.4: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support — Processing Fully Linked Partial Hyper-Link from P Corresponding to an Equation Which Is Not an Instance of $\{ x = x \}$

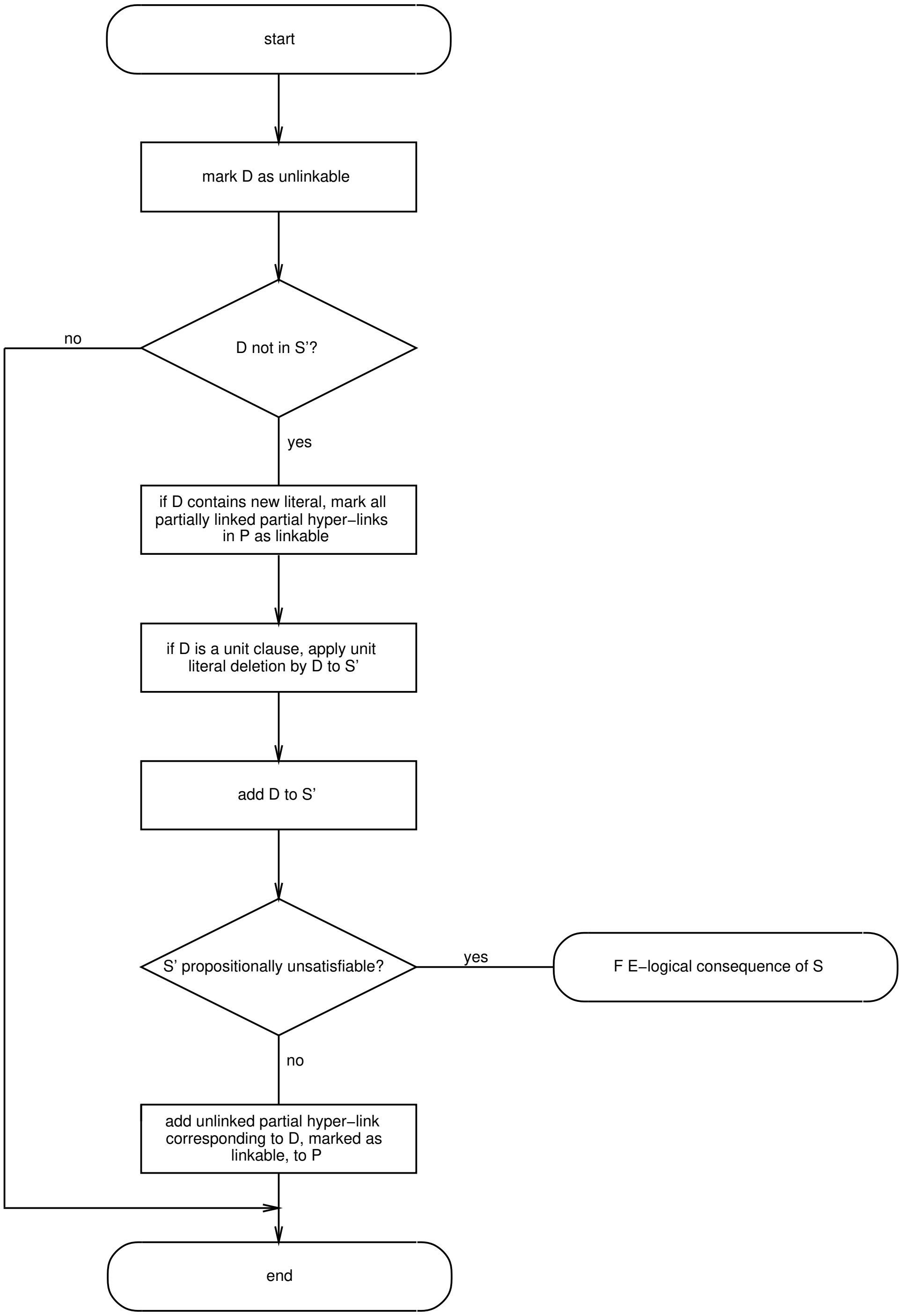

Figure 5.5: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support — Processing Fully Linked Partial Hyper-Link from P Corresponding to a Non-Equation or an Instance of $\{ x = x \}$

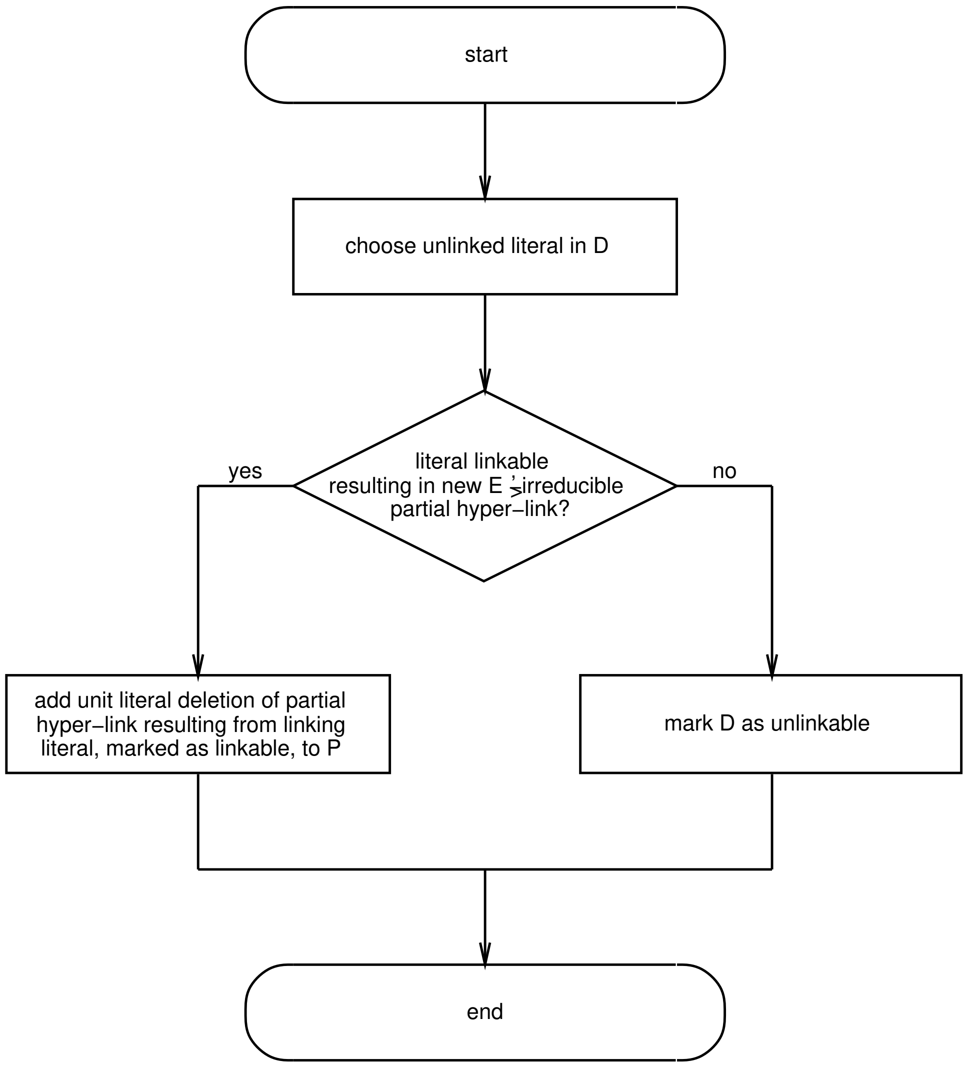

Figure 5.6: Smallest Instance First Hyper-Linking Proof Procedures with Unfailing Completion Equality Support — Processing Partially Linked Partial Hyper-Link from P

Definition 5.3 Let P' be a smallest instance first hyper-link proof procedure with unfailing equality support defined by the termination ordering $>$, specialization ordering $\rhd$, and hyper-link ordering $\succeq$, P'' the algorithm defined by removing the propositional unsatisfiability tests from P', S a finite set of closed formulas, F a closed formula, and $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ a sequence of unfailing completion equality support tuples. We say $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ is the *unfailing completion equality support sequence* of P' on (S, F) if

(a) N_0 is E-mod$(S_s) \cup \{\{x = x\}, \{x \neq y, x \neq z, y = z\}\}$ where S_s is the Skolemization of S and $\sim F$.
(b) E_0, E'_0, N'_0, S'_0 and P_0 are empty.
(c) For $i \geq 1$, E_i, E'_i, N_i, N'_i, S'_i, and P_i are E, E', N, N', S', and P, respectively, after processing of the i^{th} $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P during the execution of P'' on (S, F). ∎

Lemma 5.1 Let E be a set of equations, $>$ a termination ordering, and C and D clauses such that D is the result of a single application of $E_>$-narrowing to C. Then $E \cup \{C\} \models_= D$.

Proof: Let p be the position in C and $\{l = r\}$ the equation in E such that $C[t]_p$ and D is $C\theta[l\theta \rightarrow r\theta]_p$ (recall that $C\theta[l\theta \rightarrow r\theta]_p$ denotes the clause obtained by replacing the term $l\theta$ at position p by the term $r\theta$) where θ is a most general unifier of t and l.

Assume $E \cup \{C\}$ does not have a normal model. Then $E \cup \{C\} \models_= D$.

Assume $E \cup \{C\}$ has a normal model. Let M be an arbitrary normal model of $E \cup \{C\}$. Thus, M is a normal model of $C\theta$. Since $\{l = r\} \in E$, M is a normal model of $\{l\theta = r\theta\}$. This means that $l\theta$ and $r\theta$ have the same meaning in M. So, M is a normal model of $C\theta[l\theta \rightarrow r\theta]_p$. Since M is an arbitrary normal model of $E \cup \{C\}$, $E \cup \{C\} \models_= D$. ∎

Lemma 5.2 Let S be a set of clauses, C a clause, $C\theta$ an instance of C, and D the unit literal deletion of $C\theta$ in S. Then, $S \cup \{C\} \models D$.

Proof: Since $C\theta$ is an instance of C, $S \cup \{C\} \models C\theta$. Thus, by the soundness of unit literal deletion (Theorem 2.50), $S \cup \{C\} \models D$. ∎

Lemma 5.3 Let S be a finite set of closed formulas, F a closed formula, S_s the Skolemization of S and $\sim F$, P' a smallest instance first hyper-link proof procedure with unfailing completion equality support, and $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ the unfailing completion equality support sequence of P' on (S, F). Then, for all $i \geq 0$, $S_s \models_= E_i$, $S_s \models_= E'_i$, $S_s \models_= N_i$, $S_s \models_= N'_i$, $S_s \models_= S'_i$, and $S_s \models_= P_i$.

Proof: Let $>$ be the termination ordering of P' and $\succeq$ the hyper-link ordering of P'. We show by induction that for all $i \geq 0$, $S_s \models_= E_i$, $S_s \models_= E'_i$, $S_s \models_= N_i$, $S_s \models_= N'_i$, $S_s \models_= S'_i$, and $S_s \models_= P_i$.

(*Base Step*) From Definition 5.3, N_0 is E-mod$(S_s) \cup \{ \{ x = x \}, \{ x \neq y, x \neq z, y = z \} \}$ and E_0, E'_0, N'_0, S'_0, and P_0 are empty. Thus, by Theorem 2.20, $S_s \models_= N_0$. Also, $S_s \models_= E_0$, $S_s \models_= E'_0$, $S_s \models_= N'_0$, $S_s \models_= S'_0$, and $S_s \models_= P_0$.

(*Induction Step*) Assume, as the induction hypothesis, $S_s \models_= E_n$, $S_s \models_= E'_n$, $S_s \models_= N_n$, $S_s \models_= N'_n$, $S_s \models_= S'_n$, and $S_s \models_= P_n$.

Assume that the $n + 1^{\text{st}}$ $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P is an equation D from E. This means than E_{n+1} is a subset of $E_n \cup \mathrm{EP}_>(E'_{n+1})$, E'_{n+1} is the unfailing completion reduction of $E'_n \cup \{ D' \}$ where D' is an $(E'_n)_>$-normal form of D, N_{n+1} is a subset of $N_n \cup A$ where A is the result of all single applications of $(E'_{n+1} - E'_n)_>$-narrowing on N'_{n+1}, N'_{n+1} is a subset of N'_n, S'_{n+1} is a subset of S'_n, and P_{n+1} is a subset of P_n. By the induction hypothesis, $S_s \models_= N'_{n+1}$, $S_s \models_= S'_{n+1}$, and $S_s \models_= P_{n+1}$. By the induction hypothesis and Theorem 2.35, $S_s \models_= E'_n \cup \{ D \}$. By $S_s \models_= E'_n \cup \{ D \}$ and the soundness of the unfailing completion inference rules (Theorem 2.40), $S_s \models_= E'_{n+1}$. By the induction hypothesis, $S_s \models_= E'_{n+1}$, and the soundness of the unfailing completion rules (Theorem 2.40), $S_s \models_= E_n \cup \mathrm{EP}_>(E'_{n+1})$. Thus, $S_s \models_= E_{n+1}$. By $S_s \models_= N'_{n+1}$, $S_s \models_= E'_{n+1}$, and Lemma 5.1, $S_s \models_= A$. By the induction hypothesis and $S_s \models_= A$, $S_s \models_= N_{n+1}$.

Assume that the $n+1^{\text{st}}$ $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P is a clause D from N. This means that E_{n+1} is E_n, E'_{n+1} is E'_n, N_{n+1} is $N_n \cup A$ where A is the result of all single applications of $(E'_n)_>$-narrowing on D, N'_{n+1} is $N'_n \cup \{ D \}$, S'_{n+1} is $S'_n \cup \{ D' \}$ where D' is the unit literal deletion of D, and P_{n+1} is $P_n \cup \{ D' \}$. By the induction hypothesis, $S_s \models_= E_{n+1}$, $S_s \models_= E'_{n+1}$, and $S_s \models_= N'_{n+1}$. By Lemma 5.2, $S_n \cup \{ D \} \models D'$. By the induction hypothesis and $S_n \cup \{ D \} \models D'$, $S_s \models_= S'_{n+1}$, and $S_s \models_= P_{n+1}$. By the induction hypothesis and Lemma 5.1, $S_s \models_= A$. By the induction hypothesis and $S_s \models_= A$, $S_s \models_= N_{n+1}$.

Assume that the $n+1^{\text{st}}$ $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P is a fully linked partial hyper-link D from P corresponding to an equation which is not an instance of $\{ x = x \}$. This means that E_{n+1} is $E_n \cup \{ D' \}$ where D' is the equation corresponding to D, E'_{n+1} is E'_n, N_{n+1} is N_n, N'_{n+1} is N'_n, S'_{n+1} is S'_n, and P_{n+1} is $P_n - \{ D \}$. By the induction hypothesis, $S_s \models_= E_{n+1}$, $S_s \models_= E'_{n+1}$, $S_s \models_= N_{n+1}$, $S_s \models_= N'_{n+1}$, $S_s \models_= S'_{n+1}$, and $S_s \models_= P_{n+1}$.

Assume that the $n+1^{\text{st}}$ $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P is a fully linked partial hyper-link D from P corresponding to a non-equation or an instance of $\{ x = x \}$. This means that E_{n+1} is E_n, E'_{n+1} is E'_n,N_{n+1} is N_n, N'_{n+1} is N'_n, S'_{n+1} is $A \cup \{ D \}$ where A is the unit literal deletion of S'_n by D, and P_{n+1} is $P_n \cup \{ D' \}$ where D' is the unlinked partial hyper-link corresponding to D. By the induction hypothesis, $S_s \models_= E_{n+1}$, $S_s \models_= E'_{n+1}$, $S_s \models_= N_{n+1}$, $S_s \models_= N'_{n+1}$, and $S_s \models_= P_{n+1}$. By Lemma 5.2, $S'_n \cup \{ D \} \models A$. By the induction hypothesis and $S'_n \cup \{ D \} \models A$, $S_s \models_= S'_{n+1}$.

Assume that the $n+1^{\text{st}}$ $\succeq$-smallest unprocessed equation in E, unprocessed clause in N, or linkable partial hyper-link in P is a partially linked partial hyper-link D from P. This means that E_{n+1} is E_n, E'_{n+1} is E'_n,N_{n+1} is N_n, N'_{n+1} is N'_n, S'_{n+1} is S'_n, and P_{n+1} is $P_n \cup \{ D' \}$ where D' is the unit literal deletion of an instance of D in S_n. By the induction hypothesis, $S_s \models_= E_{n+1}$, $S_s \models_= E'_{n+1}$, $S_s \models_= N_{n+1}$, $S_s \models_= N'_{n+1}$, and $S_s \models_= S'_{n+1}$. By Lemma 5.2, $S_n \cup \{ D \} \models D'$. By the induction hypothesis and $S_n \cup \{ D \} \models D'$, $S_s \models_= P_{n+1}$. ∎

Theorem 5.4 (Soundness of Smallest Instance First Hyper-Linking with Unfailing Completion Equality Support) Every smallest instance first hyper-link proof procedure with unfailing completion equality support is sound.

Proof: Let P' be an arbitrary smallest instance first hyper-link proof procedure with unfailing completion equality support and F an arbitrary closed formula such that $\vdash_{P'} F$. Let S_s be the Skolemization of $\sim F$ and $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ the unfailing completion equality support sequence of P' on $(\{\}, F)$. Since $\vdash_{P'} F$, there exists an n such that S_n is propositionally unsatisfiable. By Lemma 5.3, $S_s \models_= S_n$. Thus,

S_s is E-unsatisfiable. So, by Theorem 2.9, $\sim F$ is E-unsatisfiable. Thus, by Theorem 2.8, $\models_= F$. ■

Definition 5.4 Let P' be a smallest instance first hyper-link proof procedure with unfailing equality support, S a set of closed formulas, F a closed formula, and $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ the unfailing completion equation support sequence of P' on (S, F). The set $(E')^\infty$ of *persisting equations* of $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ is $\bigcup_{i \geq 0} \bigcap_{j \geq i} E'_j$. ■

Lemma 5.5 Let P' be a smallest instance first hyper-link proof procedure, $>$ the termination ordering of P', S a finite set of closed formulas, F a closed formula, $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ the unfailing completion equality support sequence of P' on (S, F), and $(E')^\infty$ the set of persisting equations of $(E_0, E'_0, N_0, N'_0, S'_0, P_0), (E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$. Then $((E')^\infty)_>$ is ground canonical.

Proof: Let P'' the algorithm defined by removing the propositional unsatisfiability tests from P' and $(E''_0, R_0), (E''_1, R_1), \ldots$ be the unfailing completion sequence generated during the execution of P'' on (S, F). Since P'' stores the results of all application of unfailing completion inference rules in E', $(E')^\infty$ and $E^\infty \cup (R^\infty)^=$ are identical. Since P'' eventually processes all equations in E, $\mathrm{EP}_>((E')^\infty) \subseteq \bigcup E''_i$. Thus, by Theorem 2.43, $((E')^\infty)_>$ is ground canonical. ■

Lemma 5.6 Let $E_>$ be a ground canonical ordered rewriting system, H an unsatisfiable set of ground clauses, and H' the $E_>$-normal form of H. Then H' is unsatisfiable.

Proof: Assume, towards a contradiction, that H' is satisfiable. Let M' be a model of H'. Since H' is a set ground clauses, there exists a literal L'_i in every clause C'_i of H' which is true in M'. Let M be the set of literals L_i from every clause C_i in H such that some L'_i is the $E_>$-normal form of L_i. Since H is unsatisfiable, there exist contradictory literals L_i and L_j in M. Since $E_>$ is ground canonical, L'_i and L'_j are contradictory, thus contradicting the definition of M' as a model of H'. Therefore, H' is unsatisfiable. ■

Lemma 5.7 Let E be a set of equations, $\{l = r\}$ an equation in E, $>$ a termination ordering, C, C', D, and D' clauses, and p a position in C such that D is an instance of C,

C' is the $E_>$-narrowing of C at p using $\{ l = r \}$, and D' is the $E_>$-rewrite of D at p using $\{ l = r \}$. Then D' is an instance of C'.

Proof: Let θ be the substitution such that D is $C\theta$ and t the subterm of C at p. This means that C' is $C\gamma[l\gamma \to r\gamma]_p$ (recall that $C\gamma[l\gamma \to r\gamma]_p$ denotes the term obtained by replacing the term $l\gamma$ at position p by $r\gamma$) where γ is a most general unifier of l and t and D' is $D[l\sigma \to r\sigma]_p$ where $t\theta$ is $l\sigma$. We note that $l\sigma$ is an instance of $l\gamma$. Thus, $r\sigma$ is an instance of $r\gamma$. Let γ' be the subset of γ whose bindings map variables in t. We note that θ is $\gamma'\mu$ for some substitution μ. Thus, $C\theta$ is an instance of $C\gamma'$. So, D is an instance of $C\gamma$. Therefore, D' is an instance of C'. ∎

Lemma 5.8 Let H be an unsatisfiable set of ground clauses and H' the unit literal deletion of H. Then H' is unsatisfiable.

Proof: Assume, towards a contradiction, that H' is satisfiable. Let M be a model of H'. We note that H' is simply H with zero or more literals deleted. Thus, M is also a model of H. This contradicts the definition of H as an unsatisfiable set of ground clauses. Therefore, H' is unsatisfiable. ∎

Theorem 5.9 (Completeness of Smallest Instance First Hyper-Linking with Unfailing Completion Equality Support) Every smallest instance first hyper-link proof procedure with unfailing completion equality support whose termination ordering has the full invariance property is logically complete.

Proof: Let P' be an arbitrary smallest instance first hyper-link proof procedure with unfailing completion equality support whose termination ordering $>$ has the full invariance property, F an arbitrary E-valid closed formula, $(E_0, E'_0, N_0, N'_0, S'_0, P_0)$, $(E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$ the unfailing completion equality support sequence of P' on $(\{\}, F)$, $(E')^\infty$ the set of persisting equations of $(E_0, E'_0, N_0, N'_0, S'_0, P_0)$, $(E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$, and S_s the Skolemization of $\sim F$. By Theorem 2.8, S_s is E-unsatisfiable. Thus, by Theorem 2.20, N_0 is unsatisfiable. So, by the Skolem-Herbrand-Gödel Theorem (Theorem 2.17), there exists an unsatisfiable finite subset H of ground instances of N_0. By Lemma 5.5, $((E')^\infty)_>$ is ground canonical. Let H' be the $((E')^\infty)_>$-normal form of H. By Lemma 5.6, H' is unsatisfiable.

Let D be an arbitrary clauses in H, $D_0, D_1, \ldots, D_n$ a sequence of clauses such that

(a) D_0 is D.
(b) D_n is the $((E')^\infty)_>$-normal form of D.
(c) For $0 \leq i < n$, D_{i+1} is the $((E')^\infty)_>$-rewrite of D_i at position p_i using equation $\{ l_i = r_i \}$ from $(E')^\infty$.

and $C_0, C_1, \ldots, C_n$ a sequence of clauses such that

(a) C_0 is in N_0.
(b) D is an instance C_0.
(c) For $0 \leq i < n$,
  (i) If p_i does not occur in C_i, C_{i+1} is C_i.
  (ii) If $C_i[x_i]_{p_i}$ with x_i a variable, C_{i+1} is C_i.
  (iii) If $C_i[t_i]_{p_i}$ with t_i not a variable, C_{i+1} is the $((E')^\infty)_>$-narrowing of C_i at p_i using $\{ l_i = r_i \}$.

We show by induction that for $0 \leq i \leq n$, D_i is an instance of C_i.

(*Base Step*) By definition, D_0 is an instance of C_0.

(*Induction Step*) Assume, as the induction hypothesis, that D_m is an instance of C_m. Assume that p_i does not occur in C_i. Thus, D_{m+1} is $D_m\theta_{m+1}$ and C_{m+1} is C_m. So, D_{m+1} is an instance of C_{m+1}. Assume that $C_i[x_i]_{p_i}$ with x_i a variable. Thus, D_{m+1} is $D_m\theta_{m+1}$ and C_{m+1} is C_m. So, D_{m+1} is an instance of C_{m+1}. Assume that $C_i[t_i]_{p_i}$ with t_i not a variable. Thus, by the induction hypothesis and Lemma 5.7, D_{m+1} is an instance of C_{m+1}.

We note that every C_i is eventually added to N'. Thus, since D is an arbitrary clause in H, there exists an i such that H' is a finite subset of ground instances of N_i'. Let H'' be the unit literal deletion of H'. By Lemma 5.8, H'' is unsatisfiable. Since each clause in H'' is $((E')^\infty)_>$-irreducible and since $>$ has the full invariance property, there exists a $j \geq i$ such that H'' is a finite subset of ground instances of S_j. Since H'' is unsatisfiable, S_j' is unsatisfiable. Also, since $>$ has the full invariance property, every hyper-link instance in an $((E')^\infty)_>$-irreducible set is $((E')^\infty)_>$-irreducible. Thus, by the completeness of smallest instance first hyper-linking with unit literal deletion[20], there exists a $k \geq j$ such that S_k' is propositionally unsatisfiable. Thus, running P' on $(\{\}, F)$ terminates with an indication of "F is an E-logical consequence of S". Therefore, $\vdash_{P'} F$. ∎

[20]The completeness of smallest instance first hyper-linking with unit literal deletion is a straightforward extension of the completeness of smallest instance first hyper-linking (Theorem 4.5).

One of the reasons smallest instance first hyper-linking with unfailing completion equality support is complete is that we are able to show that the set H'' of ground clauses is unsatisfiable. In smallest instance first hyper-linking with unfailing completion equality support, we use Brand's E-modification to make the set S_s of clauses, and thus H'', unsatisfiable. However, it is not necessary that we use Brand's E-modification. We could, for example, simply add the equality axioms to S_s. In fact, if we know that S_s is unsatisfiable, rather than just E-unsatisfiable, we do not have to modify or extend S_s at all; that is, we could simply initialize N to S_s.

In particular, we are able show that if S_s is E-unsatisfiable and all positive equational literals occur in equational Horn clauses, initializing N to $S_s \cup \{ x = x \}$ is sufficient to make H''' unsatisfiable where H''' is H'' with zero or more literals deleted by unit literal deletion. This includes the common case where all positive equational literals occur as equations. It is interesting to note that when S_s contains only equations and inequations, smallest instance first hyper-linking with unfailing completion equality support reduces to simply performing unfailing completion on the equations, applying narrowing (ordered rewriting) to the inequations, and searching for a unit conflict.

Definition 5.5 Let C be a Horn clause. We say C is an *equational Horn clause* if all literals in C are equational literals. ■

Lemma 5.10 Let $E_>$ be an ordered rewriting system, H an E-unsatisfiable set of clauses, and H' the $E_>$-normal form of H. Then H' is E-unsatisfiable.

Proof: Assume, towards a contradiction, that H' is E-satisfiable. Let M be a normal model of H'. Let C' be an arbitrary clause in H'. M is a normal model of C'. Let C be a clause in H such that $C \rightarrow_{E_>} C'$. By Theorem 2.35, M is a normal model of C. Since C' is an arbitrary clause in C, M is a normal model of H. Thus, H is E-satisfiable. This contradicts the definition of H as an E-unsatisfiable set of clause. Therefore, H' is E-unsatisfiable. ■

Theorem 5.11 Let P' be a smallest instance first hyper-link proof procedure with unfailing completion equality support whose termination ordering $>$ has the full invariance property, F an E-valid closed, S_s the Skolemization of $\sim F$, P'' the algorithm defined by modifying P so that N is initially set to S_s, $(E_0, E_0', N_0, N_0', S_0', P_0), (E_1, E_1', N_1, N_1', S_1', P_1), \ldots$ the unfailing completion sequence of P'', and $(E')^{\infty}$ the persisting equations of $(E_0, E'_0, N_0, N'_0, S'_0, P_0)$, $(E_1, E'_1, N_1, N'_1, S'_1, P_1), \ldots$. If all positive equational literals in S_s occur in equational Horn clauses, then $\vdash_{P''} F$.

Proof: Assume all positive equational literals in S_s occur in equational Horn clauses. By Theorem 2.8, S_s is E-unsatisfiable. So, by the Skolem-Herbrand-Gödel Theorem for Equality (Theorem 2.18), there exists an E-unsatisfiable finite subset H of ground instances of S_s. Since N_0 is S_s, G is an E-unsatisfiable finite subset of ground instances of N_0. By Lemma 5.5, $((E')^{\infty})_{>}$ is ground canonical. Let H' be the $((E')^{\infty})_{>}$-normal form of H. By Lemma 5.10, H' is E-unsatisfiable.

Let D be an arbitrary clauses in H, $D_0, D_1, \ldots, D_n$ a sequence of clauses such that

(a) D_0 is D.
(b) D_n is the $((E')^{\infty})_{>}$-normal form of D.
(c) For $0 \leq i < n$, D_{i+1} is the $((E')^{\infty})_{>}$-rewrite of D_i at position p_i using equation $\{ l_i = r_i \}$ from $(E')^{\infty}$.

and $C_0, C_1, \ldots, C_n$ a sequence of clauses such that

(a) C_0 is in N_0.
(b) D is an instance C_0.
(c) For $0 \leq i < n$,
  (i) If p_i does not occur in C_i, C_{i+1} is C_i.
  (ii) If $C_i[x_i]_{p_i}$ with x_i a variable, C_{i+1} is C_i.
  (iii) If $C_i[t_i]_{p_i}$ with t_i not a variable, C_{i+1} is the $((E')^{\infty})_{>}$-narrowing of C_i at p_i using $\{ l_i = r_i \}$.

We show by induction that for $0 \leq i \leq n$, D_i is an instance of C_i.

(*Base Step*) By definition, D_0 is an instance of C_0.

(*Induction Step*) Assume, as the induction hypothesis, that D_m is an instance of C_m. Assume that p_i does not occur in C_i. Thus, D_{m+1} is $D_m\theta_{m+1}$ and C_{m+1} is C_m. So, D_{m+1} is an instance of C_{m+1}. Assume that $C_i[x_i]_{p_i}$ with x_i a variable. Thus, D_{m+1} is $D_m\theta_{m+1}$ and C_{m+1} is C_m. So, D_{m+1} is an instance of C_{m+1}. Assume that $C_i[t_i]_{p_i}$ with t_i not a variable. Thus, by the induction hypothesis and Lemma 5.7, D_{m+1} is an instance of C_{m+1}.

We note that every C_i is eventually added to N'. Thus, since D is an arbitrary clause in H, there exists an i such that H' is a finite subset of ground instances of N'_i. Let H'' be the unit literal deletion of H'. Since H'' is simply H' with zero or more literals deleted, H'' is

E-unsatisfiable and $((E')^{\infty})_{>}$-irreducible. Since each clause in H'' is $((E')^{\infty})_{>}$-irreducible and since $>$ has the full invariance property, there exists a $j \geq i$ such that H'' is a finite subset of ground instances of S'_j.

Let $\{ s = t \}$ be an arbitrary equation in H''. Assume, towards a contradiction, that s and t differ, We note that $\{ s = t \}$ is eventually add to E' during the execution of P'' on $(\{ \}, F)$. So, since s and t differ, $\{ s = t \}$ is $((E')^{\infty})_{>}$-reducible. This contradicts the statement that H'' is $((E')^{\infty})_{>}$-irreducible. Therefore, every equation in H'' is an instance of $\{ x = x \}$.

Since N contains $\{ x = x \}$ and $\{ x = x \}$ is $((E')^{\infty})_{>}$-irreducible, $\{ x = x \}$ is eventually added to S'. Thus, there exists a $k \geq j$ such that $\{ x = x \} \in S'_k$. Since $\{ x = x \} \in S'_k$, there exists an $l \geq k$ where all literals of the form $s \neq s$ have been deleted by unit literal deletion with $\{ x = x \}$. Let H''' be the set of clauses in S'_l corresponding to the clauses in H''. We note that H''' is H'' with zero or more literals deleted by unit literal deletion.

Since H'' is E-unsatisfiable, H''' is E-unsatisfiable. Assume, towards a contradiction, that H''' is satisfiable. Let M be a model of H'''. Let $C_0, C_1, \ldots, C_n$ be the clauses of H''' and $L_0, L_1, \ldots, L_n$ literals in $C_0, C_1, \ldots, C_n$ which are true in M. Since all equations in H'', and thus all equations in H''', are instances of $\{ x = x \}$, each L_i of the form $\{ s = t \}$ where s and t differ occurs in a non-unit clause C_i. Let $L'_0, L'_1, \ldots, L'_n$ be the literals $L_0, L_1, \ldots, L_n$ where each L_i of the form $s = t$ is replaced by another literal in C_i. Since all positive equational literals occur in equational Horn clauses, each L_i of the form $s = t$ where s and t differ is replaced by a negative equational literal. Thus, $L'_0, L'_1, \ldots, L'_n$ contains no positive equational literal and no negative equational literals of form $s \neq s$. Thus, the literals $L'_0, L'_1, \ldots, L'_n$ are true in some normal model M' of H'''. So, H''' is E-satisfiable. This contradicts the statement that H''' is E-unsatisfiable. Therefore, H''' is unsatisfiable.

Since H''' is unsatisfiable, S'_l is unsatisfiable. Also, since $>$ has the full invariance property, every hyper-link instance in an $((E')^{\infty})_{>}$-irreducible set is $((E')^{\infty})_{>}$-irreducible. Thus, by the completeness of smallest instance first hyper-linking with unit literal deletion[21], there exists an $m \geq l$ such that S'_m is propositionally unsatisfiable. Thus, running P'' on

[21] The completeness of smallest instance first hyper-linking with unit literal deletion is a straightforward extension of the completeness of smallest instance first hyper-linking (Theorem 4.5).

$(\{\,\}, F)$ terminates with an indication of "F is an E-logical consequence of S". Therefore, $\vdash_{P''} F$. ∎

In smallest instance first hyper-linking with unfailing completion equality support, the sets S' and P contain only $E'_>$-irreducible clauses and partial hyper-links, as only $E'_>$-irreducible hyper-link instances are needed for completeness. $E'_>$-reducible clauses and partial hyper-links are never added to, or when E' changes, are deleted from S' and P. However, one could instead rewrite the $E'_>$-reducible clauses and partial hyper-links to some $E'_>$-normal form. Furthermore, in looking at the proofs of Theorems 5.9 and 5.11, we note that narrowing is only needed to ensure that H' is a finite subset of ground instances of some N'_i. Thus, since every clause in H' is $((E')^\infty)_>$-irreducible, one could rewrite the clauses in the sets N and N' to some $E'_>$-normal form and still maintain completeness. We believe that applying $E'_>$-ordered rewriting to the sets N, N', S', and P should result, in most cases, in shorter proof times.

Chapter 6

Smallest Instance First Hyper-Linking Implementation and Results

We have developed a preliminary implementation of smallest instance first hyper-linking which we refer to as CLIN-2. CLIN-2 provides two variants of the hyper-link ordering presented in Figure 6.1: one where the partial hyper-link size is the maximum size of a literal in the partial hyper-link and one where the partial hyper-link size is the sum of the sizes of the literals in the partial hyper-link. In both variants of the hyper-link ordering, literal size is the number of predicate symbols, function symbols, and variables in the literal. When comparing the literals or unlinked literals of the partial hyper-links C and D, the literals are sorted using the ordering on literals presented in Figure 6.2 and the sorted literals compared pairwise in order. The literal ordering recursively uses the ordering on terms presented in Figure 6.3 to compare predicate and function arguments, where the size of a term is the number of function symbols and variables in the term and $\mathit{Variable\_Ordering}(x, y)$ returns **true** if the variable x was defined to CLIN-2 before the variable y and **false** otherwise.

Note that special care is taken to ensure that when comparing the literals or unlinked literals of a partial hyper-link, variant literal sets are considered equal. This can be be somewhat tricky. For example, consider the sorted literals sets $\{\, p(w, x), p(x, w), q(w) \,\}$ and $\{\, p(y, z), p(z, y), q(z) \,\}$. These two sets are variants of each other under the substitution

$\{w \mapsto z, x \mapsto y\}$. Note that deciding if two literal sets are variant cannot be done by a straight pairwise comparison of the literals; all variant literals must be compared. In order to avoid the need to compare every literal in the first literal set with every literal in the second literal set, we note that the literal ordering presented in Figure 6.2 orders variant literals next to one another.

CLIN-2 allows the user to choose which instances are generated: all hyper-link instances, backward supported hyper-link instances, forward supported hyper-link instances, or UR resolution instances. It does not support the generation of user supported hyper-link instances. CLIN-2 includes support for tautology deletion, a simplified form of unit subsumption, and a simplified form of unit literal deletion. It does not include support for ground unit literal generation, proper subsumption, predicate replacement, automated clause splitting, Brand's E-modification, or automated group detection. Most importantly, CLIN-2 does not include unfailing completion equality support.

Simplified unit subsumption differs from the unit subsumption described in Section 4.5 in that simplified unit subsumption only performs unit subsumption on clauses; no partial hyper-links are deleted by unit subsumption. Simplified unit literal deletion differs from the unit literal deletion described in Section 4.6 in that simplified unit literal deletion only performs unit literal deletion on clauses and embedded unit literal deletion on partial hyper-links; general unit literal deletion on partial hyper-links is not done. Furthermore, unit links are not performed when partial hyper-links are added to the partial hyper-link set.

Round-by-round hyper-linking performs a propositional satisfiability test only once per round, while smallest instance first hyper-linking is required to perform a propositional satisfiability test each time a clause is added to the clause set. This poses a serious problem for smallest instance first hyper-linking since performing a propositional satisfiability test each time a clause is added to the clause set is very inefficient for all but the most trivial theorems. We address this inefficiency by associating a ground model with the clause set.

Definition 6.1 Let S be a set of clauses, L a literal in some clause of S, and \$ an individual constant not occurring in S. The *ground* of L, denoted by $\mathrm{Gr}(L)$, is the literal obtained by replacing each variable in L by \$. ∎

```
Algorithm Hyper_Link_Ordering(C, D)
  Input
    C:  partial hyper-link
    D:  partial hyper-link
  Output
    true:  C < D
    false:  C ≥ D

  begin
    if sizeof(C) < sizeof(D) then return true
    else if sizeof(C) > sizeof(D) then return false
    else
      if number_of_unlinked_literals(C) < number_of_unlinked_literals(D)
        then return true
      else if number_of_unlinked_literals(C) > number_of_unlinked_literals(D)
        then return false
      else
        if number_of_literals(C) < number_of_literals(D) then return true
        else if number_of_literals(C) > number_of_literals(D) then return false
        else
          if the literals of C are less than the literal of D then return true
          else if the literals of C are are greater than the literals D then return false
          else
            if the unlinked literals of C are less than the unlinked literals of D
              then return true
            else return false
  end
```

Figure 6.1: Algorithm *Hyper_Link_Ordering*

```
Algorithm Literal_Ordering(L, M)
  Input
    L:  literal
    M:  literal
  Output
    true:   L < M
    false:  L ≥ M

  begin
    if sizeof(L) < sizeof(M) then return true
    else if sizeof(L) > sizeof(M) then return false
    else
      if sign(L) is positive and sign(M) is negative then return true
      else if sign(L) is negative and sign(M) is positive then return false
      else
        if arity(L) < arity(M) then return true
        else if arity(L) > arity(M) return false
        else
          if predicate_symbol(L) < predicate_symbol(M) then return true
          else if predicate_symbol(L) > predicate_symbol(M) then return false
          else
            for i := 1 to arity(L)
              if Term_Ordering(argument(L, i), argument(M, i)) then return true
              else if Term_Ordering(argument(M, i), argument(L, i))
                then return false
            return false
  end
```

Figure 6.2: Algorithm *Literal_Ordering*

```
Algorithm Term_Ordering(s, t)
  Input
    s:  term
    t:  term
  Output
    true:  s < t
    false:  s ≥ t

  begin

    if sizeof(s) < sizeof(t) then return true
    else if sizeof(s) > sizeof(s) then return false
    else
      if type(s) is variable and sign(t) is variable then return Variable_Ordering(s, t)
      else if type(s) is variable and type(t) is function then return true
      else if type(s) is function and type(t) is variable then return false
      else
        if arity(s) < arity(t) then return true
        else if arity(s) > arity(t) return false
        else
          if function_symbol(s) < function_symbol(t) then return true
          else if function_symbol(s) > function_symbol(t) then return false
          else
            for i := 1 to arity(s)
              if Term_Ordering(argument(s, i), argument(t, i)) then return true
              else if Term_Ordering(argument(t, i), argument(s, i)) then return false
            return false
  end
```

Figure 6.3: Algorithm *Term_Ordering*

Definition 6.2 Let S be a set of clauses and M a set of ground literals. We say M is ground model of S if

(a) for every clause C in S, there exists a literal L in C such that $\mathrm{Gr}(L)$ is in M.
(b) there exist no literals L_1 and L_2 in M such that L_1 and L_2 are contradictory. ∎

Theorem 6.1 Let S be a set of clauses. Then, S is propositionally satisfiable iff S has a ground model.

Proof: Directly from the definition of propositional satisfiability. ∎

CLIN-2 starts with both the clause set S and ground model M empty. When a clause C is added to S, CLIN-2 first determines if C contains a literal L such that $\mathrm{Gr}(L)$ is in M. If C does contain such a literal, M is a ground model of $S \cup \{ C \}$. In this case, M is not modified. Otherwise, CLIN-2 determines if C contains a literal L such that $\mathrm{Gr}(L)$ does not contradict any literal in M. If C does contain such a literal, $M \cup \{ \mathrm{Gr}(L) \}$ is a ground model of $S \cup \{ C \}$. In this case, $\mathrm{Gr}(L)$ is added to M. Otherwise, CLIN-2 uses a propositional satisfiability test based on the method of Davis and Putnam[24] to find a ground model for $S \cup \{ C \}$.

Results of running CLIN and CLIN-2 on Version 1.1.3 of the TPTP Problem Library of Sutcliffe and Suttner[93,94] are presented in Tables B.1 and B.2. Table 6.1 presents a summary of the TPTP V1.1.3 problems solved by CLIN and CLIN-2 compiled by TPTP domain. Tables 6.2 and 6.3 present a comparison of CLIN and CLIN-2 compiled by TPTP domain. In comparing CLIN and CLIN-2, we say that CLIN and CLIN-2 perform the same on a given problem if

(a) The times required by CLIN and CLIN-2 to solve the problem are both less than or equal to two seconds.
(b) Both CLIN and CLIN-2 fail to solve the problem.
(c) The absolute value of the difference in the times required by CLIN and CLIN-2 to solve the problem is less than or equal to five percent of the average of the times. For example, say that CLIN solves a problem in 20.5 seconds and CLIN-2 solves the same problem in 19.5 seconds. The absolute value of the difference in times in 1.0 seconds and the average is 20 seconds. Since five percent of 20 is 1.0, we say that CLIN and CLIN-2 performed the same on the problem.

Domain	Propositional		Non-Propositional					Total Problems Solved	
			Forward Support			Backward Support			
			Max		Sum	Max	Sum		
	CLIN	CLIN-2	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2	CLIN	CLIN-2
ALG			001/004	001/004	001/004	000/004	000/004	001/004	001/004
ANA			000/019	000/019	000/019	000/019	000/019	000/019	000/019
BOO			005/051	002/051	000/051	001/051	000/051	005/051	002/051
CAT			029/059	022/059	017/059	009/059	015/059	029/059	024/059
CID			000/004	000/004	000/004	000/004	000/004	000/004	000/004
CIV			000/004	000/004	000/004	000/004	000/004	000/004	000/004
COL			043/105	029/105	030/105	008/105	025/105	043/105	032/105
COM			005/006	005/006	004/006	003/006	004/006	005/006	005/006
GEO			059/165	052/165	035/165	008/165	017/165	059/165	057/165
GRA	001/001	001/001						001/001	001/001
GRP			056/148	050/148	013/148	013/148	014/148	056/148	050/148
HEN			025/064	027/064	010/064	007/064	008/064	025/064	029/064
LAT			002/010	000/010	000/010	000/010	000/010	002/010	000/010
LCL	002/002	002/002	111/276	069/276	053/276	028/276	036/276	113/278	080/278
LDA			002/023	002/023	002/023	000/023	000/023	002/023	002/023
MSC	007/014	005/014	012/028	011/028	008/028	007/028	005/028	019/042	016/042
NUM	001/001	001/001	031/321	023/321	023/321	010/321	015/321	032/322	025/322
PLA			005/030	005/030	002/030	008/030	008/030	005/030	008/030
PRV			007/009	003/009	002/009	003/009	002/009	007/009	003/009
PUZ	010/010	010/010	025/039	025/039	024/039	023/039	022/039	035/049	035/049
RNG			018/100	013/100	007/100	006/100	007/100	018/100	013/100
ROB			003/034	003/034	002/034	002/034	002/034	003/034	003/034
SET			067/700	019/700	015/700	017/700	016/700	067/700	023/700
SYN	084/094	077/094	267/310	238/310	233/310	279/310	283/310	351/404	362/404
TOP			003/024	003/024	003/024	004/024	004/024	003/024	004/024
Totals	105/122	096/122	776/2533	602/2533	484/2533	436/2533	483/2533	881/2655	775/2655

Table 6.1: TPTP V1.1.3 Problems Solved

Domain	CLIN Better	Same	CLIN-2 Better
GRA	0	1	0
LCL	0	2	0
MSC	4	10	0
NUM	0	0	1
PUZ	2	6	2
SYN	25	58	11
Totals	31	77	14

Table 6.2: Comparison of CLIN and CLIN-2 TPTP V1.1.3 Propositional Problems

Domain	**CLIN Better**	**Same**	**CLIN-2 Better**
ALG	0	3	1
ANA	0	19	0
BOO	5	46	0
CAT	18	30	11
CID	0	4	0
CIV	0	4	0
COL	33	70	2
COM	4	2	0
GEO	44	99	22
GRP	43	95	10
HEN	12	41	11
LAT	2	8	0
LCL	82	173	21
LDA	1	21	1
MSC	7	16	5
NUM	11	300	10
PLA	4	26	0
PRV	6	2	1
PUZ	7	21	10
RNG	18	79	3
ROB	0	33	1
SET	59	637	4
SYN	201	74	35
TOP	1	23	0
Totals	558	1826	148

Table 6.3: Comparison of CLIN and CLIN-2 TPTP V1.1.3 Non-Propositional Problems

Tables 6.4 and 6.5 present the proof types for those problems in which CLIN performed better than CLIN-2, compiled by TPTP domain. Tables 6.6 and 6.7 present the proof types for those problems which CLIN solved and CLIN-2 did not solve using forward support, maximum literal size, compiled by TPTP domain. Tables 6.8 and 6.9 present the proof types for those problems which CLIN solved and CLIN-2 did not solve using any of the four strategies, compiled by TPTP domain.

Domain	Embedded Unit Literal Deletion	Propositional Unsatisifiability Test	Simple Small Proof	Unit Literal Deletion
MSC	0	4	0	0
PUZ	0	1	0	1
SYN	0	15	0	10
Totals	0	20	0	11

Table 6.4: Proof Types for the Propositional Problems in Which CLIN Performed Better Than CLIN-2

Domain	Embedded Unit Literal Deletion	Propositional Unsatisifiability Test	Simple Small Proof	Unit Literal Deletion
BOO	0	0	5	0
CAT	0	2	16	0
COL	0	0	33	0
COM	0	0	2	2
GEO	0	0	43	1
GRA	0	0	0	0
GRP	0	1	42	0
HEN	0	0	12	0
LAT	0	0	2	0
LCL	0	0	82	0
LDA	0	0	1	0
MSC	0	5	2	0
NUM	0	0	8	3
PLA	0	0	4	0
PRV	0	0	4	2
PUZ	0	3	2	2
RNG	0	4	13	1
SET	0	1	45	13
SYN	2	2	159	38
TOP	0	0	0	1
Totals	2	18	475	63

Table 6.5: Proof Types for the Non-Propositional Problems in Which CLIN Performed Better Than CLIN-2

Domain	Embedded Unit Literal Deletion	Propositional Unsatisifiability Test	Simple Small Proof	Unit Literal Deletion
MSC	0	2	0	0
SYN	0	7	0	1
Totals	0	9	0	1

Table 6.6: Proof Types for the Propositional Problems Solved by CLIN, But Not CLIN-2 Using Forward Support Maximum Literal Size

Domain	Embedded Unit Literal Deletion	Propositional Unsatisifiability Test	Simple Small Proof	Unit Literal Deletion
BOO	0	0	3	0
CAT	0	1	9	0
COL	0	0	14	0
GEO	0	0	15	0
GRP	0	0	9	0
HEN	0	0	4	0
LAT	0	0	2	0
LCL	0	0	46	0
MSC	0	2	0	0
NUM	0	0	5	3
PRV	0	0	3	1
RNG	0	0	8	0
SET	0	1	35	12
SYN	0	2	37	1
Totals	0	6	190	17

Table 6.7: Proof Types for the Propositional Problems Solved by CLIN, But Not CLIN-2 Using Forward Support Maximum Literal Size

Domain	Embedded Unit Literal Deletion	Propositional Unsatisifiability Test	Simple Small Proof	Unit Literal Deletion
MSC	0	2	0	0
SYN	0	7	0	1
Totals	0	9	0	1

Table 6.8: Proof Types for the Propositional Problems Solved by CLIN, But Not CLIN-2 Using Any of the Four Strategies

Domain	**Embedded Unit Literal Deletion**	**Propositional Unsatisifiability Test**	**Simple Small Proof**	**Unit Literal Deletion**
BOO	0	0	3	0
CAT	0	1	9	0
COL	0	0	11	0
GEO	0	0	12	0
GRP	0	0	9	0
HEN	0	0	4	0
LAT	0	0	2	0
LCL	0	0	38	0
MSC	0	2	0	0
NUM	0	0	5	2
PRV	0	0	3	1
RNG	0	0	8	0
SET	0	1	31	12
SYN	0	2	0	0
Totals	0	6	135	15

Table 6.9: Proof Types for the Propositional Problems Solved by CLIN, But Not CLIN-2 Using Any of the Four Strategies

Chapter 7

Conclusion

In this dissertation, we have presented a number of reasons why round-by-round hyper-linking of Lee and Plaisted[62,63] is not particularly well suited for the addition of special methods in support of equality. We offered smallest instance first hyper-linking as an alternative to round-by-round hyper-linking. Unfailing completion equality support was developed (although not implemented) as a sound and logically complete method for adding equality support to smallest instance first hyper-linking. We developed a preliminary implementation of smallest instance first hyper-linking in an automated theorem prover called CLIN-2 and compared it to Lee's implementation of round-by-round hyper-linking, CLIN, on the entire set of TPTP V1.1.3 problems.

While in general CLIN performed better than CLIN-2 on the TPTP V1.1.3 problems, CLIN-2 did out perform CLIN on 14 propositional and 148 non-propositional problems. In fact, CLIN-2 quickly solved some problems which CLIN was unable to solve. For example, CLIN-2 solved GEO058-3, GEO067-2, HEN005-2, SYN002-1.010, SYN002-1.012, SYN091-1.020, and SYN096-1.020 in under 100 seconds, while CLIN was unable to solve any of them within 1,800 seconds. As CLIN-2 is currently only a preliminary implementation of smallest instance first hyper-linking, we believe that smallest instance first hyper-linking provides a viable alternative to round-by-round hyper-linking. Also the fact that smallest instance first hyper-linking was developed to address the inefficiencies encountered in round-by-round hyper-linking when adding equality support suggests that smallest instance first hyper-linking should perform even better with the introduction of unfailing completion equality support.

We now discuss some reasons why CLIN performs better than CLIN-2 on the TPTP V1.1.3 problems, concentrating primarily on ways to improve the efficiency of CLIN-2. We first discuss the importance of unit subsumption, unit literal deletion, unit subsumption, and unit resolution. We next discuss implementation differences between CLIN and CLIN-2 giving ways to improve CLIN-2 implementation. We conclude this chapter outlining future work to be added to CLIN-2.

The analysis of the TPTP runs given in Chapter 6 suggests that unit literal deletion, unit subsumption, and unit resolution are more important to smallest instance first hyper-linking than we had first realized. Sections 4.5 and 4.6 describe methods for adding unit subsumption and unit literal deletion to smallest instance first hyper-linking which are not fully implemented in CLIN-2. These methods should greatly improve the efficiency of CLIN-2 on a large number of the TPTP problems. For example, there are a number of theorems which CLIN proved by simple small proof checking and unit literal deletion which CLIN-2 did not prove. If with these methods added, CLIN-2 was able to prove these theorems, CLIN-2 would prove an additional 151 theorems for a total of 926 theorems, thus exceeding the 881 theorems proved by CLIN. Furthermore, the enhancements to unit subsumption and unit literal deletion may allow CLIN-2 to prove additional theorems which CLIN is unable to prove.

The implementations of CLIN and CLIN-2 differ greatly. CLIN is written in Prolog, while CLIN-2 is written in C++. We chose to use C++ instead of Prolog when writing CLIN-2 since C++ offers more flexibility in developing efficient data structures. Applications written in C++ should, in general, perform better than those written in Prolog. However, to exploit the performance benefits of C++, one must take great care in developing highly refined data structures in C++. Furthermore, C++ contains a number of features, such as automatic type conversion, which can degraded performance if one is not careful.

The data structures used in CLIN and CLIN-2 differ greatly. CLIN uses Prolog terms to represent terms and literals, Prolog lists to represent clauses, and the Prolog database to store sets of clauses. CLIN does not sort literals within clauses; thus, CLIN treats $\{p, q\}$ and $\{q, p\}$ as two different clauses. This could allow for an exponential number of redundant clauses; however, in practice this does not appear to cause a problem. CLIN uses a special representation for the literals within a clause which allows it to take advantage of Prolog's efficient unification without the occurs check[22].

CLIN-2 represents terms and literals as trees with arguments contained in an IBM C Set ++ ISequence[52, Chapter 24][23]. Each argument in the ISequence is an IBM C Set ++ IMgPtr[52, Chapter 36][24] to terms for efficient data sharing. Clauses, set of clauses, and sets of partial hyper-links are represented as IBM C Set ++ ISortedSets[52, Chapter 29][25]. Both processed and unprocessed partial hyper-links are maintained in the same set.

There are a number of inefficiencies in CLIN-2 which can be easily fixed:

- The arguments of terms and literals can be represented as an array of IMgPtrs rather than as an ISequence (linked list).

- Finding the first unprocessed partial hyper-link in the partial hyper-link set requires a linear search of the partial hyper-link set. This is quite inefficient for ISortedSets (AVL trees). Separating the partial hyper-link set into two sets, one for processed partial hyper-links and one for unprocessed partial hyper-links would eliminated the need to linearly search the partial hyper-link set.

- There are more efficient binary search trees than AVL tress, for example red-black trees. However, the IBM C Set ++ Collection Class does not offer such an implementation, so this would require a significant amount of programming.

---

[22]Clause are *linearized* by renaming the variables in the clause, if necessary, so that no variable occurs more than once. A *linearization substitution* is retained so that the original clause can be obtained from its linearized form. The use linearization allows Prolog's unification without the occurs check to be used when linking two literals. Full unification with the occurs check is applied to the linearization substitutions only when the two literal link using Prolog's unification.

[23]ISequences are implemented as linked lists.

[24]IMgPtrs are pointers with a reference count. When the reference count goes to zero, the object pointed to is deleted. This offers an efficient method of memory management when data sharing.

[25]ISortedSets are implemented as AVL trees.

- Maintaining a pointer to the first partial hyper-link in the unprocessed partial hyper-link set would reduce the complexity of finding the smallest unprocessed partial hyper-link from $O(\log n)$ to $O(1)$.

Both CLIN and CLIN-2 use a variant of the Davis-Putnam propositional unsatisfiability test[24]. CLIN implements two refinements to the unsatisfiability test, which when added to CLIN-2 should improve the efficiency of its unsatisfiability test:

- CLIN tests if the clause set is trivially satisfiable[26] after each case analysis step. CLIN-2 only checks if the initial clause set is trivially satisfiable.

- In the case analysis of a literal L, L is first assumed to be *true*. If the clause set is shown to be propositionally unsatisfiable, L is then assumed to be *false*. Lee[62] implemented in CLIN a refinement which he called *right cutoff*, where if in the *true* case L is not used in showing that the clause set is propositionally unsatisfiable, the *false* case is not performed. The propositionally unsatisfiability test in CLIN-2 does not include right cutoff.

Future enhancements to CLIN-2 should include

- enhancing unit subsumption and unit literal deletion as described in Sections 4.5 and 4.6
- improving the CLIN-2 data structures and low-level implementation as described earlier in this chapter
- adding equality completion support as described in Chapter 5
- adding support for ground unit literal generation, proper subsumption, predicate replacement, automated clause splitting, Brand's E-modification, and automated group detection as described in Chapter 4.

Possible future research topics include adding mathematical induction to hyper-linking, developing parallel implementation of hyper-linking, and applying hyper-linking to higher-order logics.

[26] A clause set is trivially satisfiable if it contains no positive clause (all literals positive) or no negative clause (all literals negative).

Appendix A

CLIN-2 User's Guide

CLIN-2 is invoked from the command line as follows:

```
clin-2 [-clause_size_method <method>] [-support_strategy <strategy>]
  [-tautology_deletion | -no_tautology_deletion] [-time_limit <limit>]
  [-trace | no_trace] [-unit_literal_deletion | -no_unit_literal_deletion]
  [-unit_subsumption | -no_unit_subsumption] <input_file> [<output_file>]
```

where

`-clause_size_method <method>`
: Specifies how clauses are sized where `<method>` is either `maximum_literal_size` or `sum_of_literal_sizes`. `maximum_literal_size` specifies that the size of a clause is the maximum size of the literals in the clause where the size of a literal is the number of predicate symbols, function symbols, and variables in the literal. `sum_of_literal_sizes` specifies that the size of a clause is the sum of the sizes of the literals in the clause where the size of a literal is the number of predicate symbols, function symbols, and variables in the literal. The default value is `sum_of_literal_sizes`.

`-support_strategy <strategy>`
: Specifies which hyper-link instances are generated where `<strategy>` is either `all`, `backward`, `forward`, `unit_resulting`, or `ur`. `all` specifies that all hyper-instances are generated. `backward` specifies that only backward supported hyper-link instances are generated. `forward` specifies that only backward supported hyper-link instances are generated. `unit_resulting` and `ur` specify that the unit resulting operation is used in place of the hyper-link operation. The default value is `all`.

`-tautology_deletion`
`-no_tautology_deletion`

Specifies whether the tautology deletion inference rule is used to delete tautologies. The default value is `tautology_deletion`.

`-time_limit <limit>`

Specifies a time limit in seconds where `<limit>` is a nonnegative integer. A time limit of `0` specifies that there is no time limit. The default value is `0`.

`-trace`
`-no_trace`

Specifies whether the trace information is generated. The default value is `no_trace`.

`-unit_literal_deletion`
`-no_unit_literal_deletion`

Specifies whether the unit literal deletion inference rule is used to delete literals from clauses and partial hyper-links. The default value is `unit_literal_deletion`.

`-unit_subsumption`
`-no_unit_subsumption`

Specifies whether the unit subsumption inference rule is used to delete clauses and partial hyper-links. The default value is `unit_subsumption`.

`<input_file>`

Specifies the name of the file containing the set of clauses. The clause set is specified using the following syntax:

clause_set :- (*whitespace_or_comment* | *clause*)*

whitespace_or_comment :- *whitespace* | *comment*

whitespace :- '`\t`' | '`\n`' | '`\v`' | '`\f`' | '`\r`' | '` `'

comment :- '`%`' *all_characters_except_newline** '`\n`'

clause :- ('`{`' *whitespace_or_comment** } | '`{`' *whitespace_or_comment** (*literal* *whitespace_or_comment** '`,`')* *whitespace_or_comment** *literal* *whitespace_or_comment** '`}`'

literal :- *predicate_symbol* | *predicate_symbol* *whitespace_or_comment** *arg_list* | '$\sim$' *predicate_symbol* | '$\sim$' *predicate_symbol* *whitespace_or_comment** *arg_list*

predicate_symbol :- ('`a`' | ⋯ | '`z`' | '`0`' | ⋯ | '`9`') ('`A`' | ⋯ | '`Z`' | '`a`' | ⋯ | '`z`' | '`0`' | ⋯ | '`9`')

arg_list :- '`(`' *whitespace_or_comment** (*term* *whitespace_or_comment** '`,`')* *whitespace_or_comment** *term* *whitespace_or_comment** '`)`'

term :- *variable* | *function_symbol* | *function_symbol* *whitespace_or_comment** *arg_list* *variable* :- (`A' | ⋯ | `Z') (`A' | ⋯ | `Z' | `a' | ⋯ | `z' | `0' | ⋯ | `9')

function_symbol :- (`a' | ⋯ | `z' | `0' | ⋯ | `9') (`A' | ⋯ | `Z' | `a' | ⋯ | `z' | `0' | ⋯ | `9')

`<output_file>`

Specifies the name of the output file. The default value is `<input_file>.out`.

Appendix B

CLIN and CLIN-2 TPTP V1.1.3 Results

We ran CLIN-2 on all problems in Version 1.1.3 of the TPTP Problem Library of Sutcliffe and Suttner[93,94]. Propositional problems were runs the following option:

- `-time_limit 1800`

Each non-propositional problem was run four times, one time with each of the following option settings:

- `-clause_size_method maximum_literal_size`
  `-support_strategy backward -time_limit 1800`
- `-clause_size_method sum_of_literal_sizes`
  `-support_strategy backward -time_limit 1800`
- `-clause_size_method maximum_literal_size`
  `-support_strategy forward -time_limit 1800`
- `-clause_size_method sum_of_literal_sizes`
  `-support_strategy forward -time_limit 1800`

As a comparison, we also ran CLIN on all problems in Version 1.1.3 of the TPTP Problem Library. For both propositional and non-propositional problems, the following flag settings were specified:

- `assign(support_list,[f]).`
  `set(slidepriority).`

CLIN-2 was compiled with IBM C Set ++ for AIX/6000 Version 2.1 and CLIN was compiled with ECLiPSe Prolog Version 3.4.5. CPU time usage for the CLIN runs was controlled via the csh "`limit cputime 1800`" command. All runs were performed on two IBM RiscSystem/6000 Model 550s: one having 1,024 MB of real memory and one having 512 MB of real memory[27].

Table B.1 presents the results of the propositional runs and Table B.2 presents the results of the non-propositional runs. All times are given as user CPU time in seconds. A horizontal line indicates that a proof was not found within the specified time limit of 1,800 seconds. CLIN and CLIN-2 were run on all TPTP V1.1.3 problems; however in order to save space, results are given only for those problems solved by at least one CLIN or CLIN-2 run. Also, in order to make a fairer comparison between CLIN-2 and CLIN, the CPU time for CLIN is sum of the "interp" time (time to read the clauses) and "Total-Time" (time to find the proof).

Problem	**CLIN**	**CLIN-2**
GRA001-1	0.20	0.18
LCL181-2	0.09	0.02
LCL230-2	0.10	0.04
MSC007-1.002	0.08	0.02
MSC007-1.003	0.16	0.12
MSC007-1.004	0.42	0.61
MSC007-1.005	1.74	6.86
MSC007-1.006	11.60	273.53
MSC007-1.007	112.01	———
MSC007-1.008	1,284.64	———
NUM285-1	10.06	1.05
PUZ004-1	0.24	0.23
PUZ008-2	0.98	2.22
PUZ009-1	0.29	0.19
PUZ013-1	0.35	0.46
PUZ014-1	0.45	0.50
PUZ028-3	233.36	75.42

Table B.1: CLIN and CLIN-2 Results on TPTP V1.1.3 Propositional Problems (continued)

[27]None of the runs ran out of memory, so the times are comparable between the two machines.

Problem	CLIN	CLIN-2
PUZ028-4	74.25	7.54
PUZ030-0.rm	0.02	0.00
PUZ030-2	1.38	3.28
PUZ033-1	0.31	0.26
SYN001-1.001	0.04	0.03
SYN001-1.002	0.07	0.07
SYN001-1.003	0.14	0.12
SYN001-1.004	0.29	0.60
SYN001-1.005	0.67	3.17
SYN001-1.006	1.67	19.64
SYN001-1.007	4.55	138.15
SYN001-1.008	13.03	1,107.55
SYN001-1.009	40.52	———
SYN001-1.010	139.43	———
SYN001-1.011	511.26	———
SYN003-1.002	0.14	0.10
SYN003-1.003	0.21	0.15
SYN003-1.004	0.31	0.26
SYN003-1.005	0.37	0.35
SYN003-1.006	0.43	0.43
SYN003-1.007	0.51	0.52
SYN003-1.008	0.59	0.64
SYN003-1.009	0.70	0.72
SYN003-1.010	0.76	0.82
SYN004-1.002	0.08	0.07
SYN004-1.003	0.15	0.12
SYN004-1.004	0.16	0.17
SYN004-1.005	0.23	0.28
SYN004-1.006	0.29	0.39
SYN004-1.007	0.35	0.52
SYN004-1.008	0.40	0.62
SYN004-1.009	0.48	0.76

Table B.1: CLIN and CLIN-2 Results on TPTP V1.1.3 Propositional Problems (continued)

Problem	CLIN	CLIN-2
SYN004-1.010	0.54	0.91
SYN007-1.001	0.09	0.02
SYN007-1.002	0.14	0.06
SYN007-1.003	0.54	0.25
SYN007-1.004	2.53	1.45
SYN007-1.005	12.49	7.75
SYN007-1.006	63.60	41.86
SYN007-1.007	403.70	242.78
SYN008-1	0.10	0.05
SYN010-1.002	0.13	0.08
SYN010-1.005	0.33	0.69
SYN010-1.010	0.71	4.40
SYN010-2.002	0.28	0.21
SYN010-2.005	0.92	2.19
SYN010-2.010	3.25	9.87
SYN010-3.002	0.73	0.47
SYN010-3.005	3.27	5.07
SYN010-3.010	12.84	18.16
SYN011-1	0.16	0.07
SYN028-1	0.10	0.04
SYN029-1	0.09	0.05
SYN030-1	0.16	0.08
SYN032-1	0.12	0.09
SYN040-1	0.06	0.04
SYN041-1	0.05	0.02
SYN042-1.rm	0.00	0.00
SYN043-1.rm	0.01	0.02
SYN044-1	0.09	0.09
SYN045-1	0.08	0.07
SYN046-1	0.05	0.02
SYN047-1	0.11	0.06
SYN085-1.005	0.13	0.10

Table B.1: CLIN and CLIN-2 Results on TPTP V1.1.3 Propositional Problems (continued)

Problem	**CLIN**	**CLIN-2**
SYN085-1.010	0.19	0.25
SYN085-1.020	0.39	1.22
SYN086-1.005	0.98	1.00
SYN086-1.010	4.27	5.16
SYN086-1.020	19.94	24.85
SYN087-1.005	0.76	0.70
SYN087-1.010	1.65	1.69
SYN087-1.020	3.41	3.89
SYN089-1.005	1.96	7.82
SYN089-1.010	17.03	122.70
SYN089-1.020	307.87	——
SYN090-1.005	1.82	2.77
SYN090-1.010	6.63	7.15
SYN090-1.020	30.62	17.89
SYN091-1.005	57.60	2.39
SYN091-1.010	1,264.90	12.06
SYN091-1.020	——	57.25
SYN092-1.005	25.75	1.58
SYN092-1.010	190.83	3.78
SYN092-1.020	1,466.65	8.65
SYN093-1.005	12.67	1,563.03
SYN093-1.010	1,285.30	——
SYN094-1.005	16.57	——
SYN097-1.005	1,732.24	——
SYN098-1.005	249.90	——

Table B.1: CLIN and CLIN-2 Results on TPTP V1.1.3 Propositional Problems

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
ALG002-1	335.97	74.62	907.69	—	—
BOO003-1	314.98	1,582.92	—	—	—
BOO004-1	314.97	—	—	—	—
BOO005-1	313.45	—	—	—	—
BOO006-1	331.63	—	—	—	—
BOO011-1	1.93	7.64	—	123.34	—
CAT001-4	128.28	—	—	—	—
CAT002-4	26.33	—	—	—	—
CAT003-3	1,394.76	—	—	—	—
CAT003-4	109.08	—	—	—	—
CAT004-4	187.90	—	—	—	—
CAT005-1	80.63	1,255.66	—	—	—
CAT005-4	1,411.70	114.40	—	—	—
CAT006-1	80.96	1,315.82	—	—	—
CAT006-4	353.71	374.66	1,037.85	—	—
CAT007-1	4.04	3.90	—	484.23	—
CAT007-3	0.44	270.05	22.78	0.92	1.38
CAT008-1	—	1,031.23	—	—	—
CAT011-1	3.20	—	—	—	—
CAT011-2	14.45	47.99	209.48	—	127.65
CAT011-4	256.42	90.54	620.59	—	—
CAT012-1	3.18	—	—	—	—
CAT012-3	—	—	510.92	—	40.85
CAT012-4	38.50	5.59	29.11	—	26.92
CAT013-1	3.22	—	—	—	—
CAT013-3	—	—	528.49	—	41.21
CAT013-4	15.61	4.75	31.40	—	27.21
CAT014-1	3.31	—	—	—	—
CAT014-2	14.66	10.79	94.37	—	122.98
CAT014-4	268.15	34.40	—	—	—

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
CAT015-3	1.00	2.54	2.50	2.53	2.49
CAT015-4	0.75	1.77	1.76	1.79	1.74
CAT016-3	—	292.22	62.41	302.69	111.55
CAT016-4	15.35	2.31	2.02	248.89	67.84
CAT017-3	—	291.32	35.32	302.45	111.39
CAT017-4	36.91	4.14	3.25	—	41.44
CAT018-1	17.87	283.61	—	—	—
CAT019-1	1.46	3.18	3.17	3.27	3.27
CAT019-2	0.60	1.15	2.26	1.26	1.27
CAT019-3	11.55	—	—	—	—
COL001-1	57.31	—	—	—	—
COL001-2	2.86	—	—	—	—
COL002-1	2.21	—	—	—	—
COL002-3	966.43	—	—	—	4.67
COL007-1	0.26	0.18	0.18	0.19	0.18
COL008-1	0.33	20.52	8.40	—	1.34
COL009-1	1.18	—	—	—	—
COL010-1	0.36	8.74	4.16	—	18.00
COL012-1	0.25	0.20	0.19	0.20	0.19
COL013-1	0.33	0.28	0.30	0.28	0.27
COL014-1	0.29	0.26	0.27	0.27	0.27
COL015-1	0.33	20.53	8.48	—	1.38
COL016-1	0.30	0.36	0.36	0.36	0.37
COL017-1	0.40	—	—	—	1.97
COL018-1	0.32	0.37	0.37	0.36	0.38
COL019-1	1.72	983.95	523.56	—	—
COL020-1	1.78	—	598.68	—	—
COL021-1	0.36	27.59	10.09	—	1.58
COL022-1	0.35	87.80	15.64	—	1.58
COL023-1	13.17	590.89	112.13	—	—

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
COL024-1	0.37	27.46	10.01	—	1.54
COL025-1	0.34	3.50	2.57	—	18.71
COL026-1	25.62	—	—	—	—
COL027-1	13.92	1,414.86	195.11	—	—
COL028-1	13.29	590.72	112.16	—	—
COL029-1	0.32	0.24	0.24	0.23	0.24
COL030-1	32.24	411.53	239.16	—	—
COL031-1	1.33	137.16	32.99	—	1.26
COL032-1	32.81	—	—	—	—
COL035-1	45.59	—	—	—	—
COL039-1	97.97	—	—	—	—
COL040-1	1,101.74	—	—	—	—
COL044-1	1,102.48	—	—	—	—
COL045-1	0.39	22.18	10.32	—	1.58
COL048-1	0.36	87.54	9.85	—	1.59
COL050-1	1.29	7.00	6.52	—	1.63
COL051-1	13.15	94.75	21.37	—	2.56
COL052-1	258.21	—	—	—	—
COL053-1	0.41	22.65	12.45	—	1,171.06
COL054-1	23.15	19.46	8.28	—	—
COL055-1	0.26	0.37	1.51	9.42	53.88
COL056-1	5.33	3.50	7.28	—	1,630.87
COL058-1	32.93	783.24	290.64	—	10.34
COM001-1	0.75	0.62	0.79	42.30	43.96
COM002-1	2.63	2.83	3.85	556.45	743.00
COM002-2	45.59	621.89	812.21	673.57	901.00
COM003-2	5.58	27.32	8.79	—	62.27
COM004-1	2.87	36.31	—	—	—
GEO001-2	—	1,229.74	—	—	—
GEO002-1	—	286.76	—	—	—

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
GEO002-2	———	284.76	———	———	———
GEO002-3	4.25	25.85	30.35	28.57	26.26
GEO002-4	92.03	4.43	245.61	———	———
GEO003-1	81.07	22.71	16.05	———	———
GEO003-2	62.53	22.56	16.02	———	———
GEO003-3	4.11	———	866.24	———	27.94
GEO006-3	166.50	71.43	———	215.69	———
GEO011-2	3.16	9.46	———	———	———
GEO011-3	8.74	———	———	———	———
GEO011-4	3.10	9.64	———	———	———
GEO011-5	1.74	5.02	———	———	———
GEO014-2	2.74	9.17	655.22	———	———
GEO015-2	6.86	15.83	1,798.70	———	———
GEO015-3	2.83	11.88	989.54	———	———
GEO016-2	2.72	17.93	———	———	———
GEO016-3	2.92	14.37	1,244.41	———	———
GEO017-2	72.96	18.88	———	———	———
GEO017-3	11.33	15.47	20.26	———	17.88
GEO018-2	6.86	12.31	791.21	———	———
GEO018-3	11.55	16.39	21.20	———	17.91
GEO019-2	2.77	11.37	787.87	———	———
GEO019-3	2.87	13.57	18.30	———	15.11
GEO020-2	72.64	13.27	793.56	———	———
GEO020-3	11.33	12.64	17.45	———	14.21
GEO021-2	6.59	16.84	———	———	———
GEO021-3	11.49	14.49	19.30	———	15.13
GEO022-2	78.93	37.40	———	———	———
GEO022-3	13.85	44.80	———	———	———
GEO024-2	61.62	24.72	88.57	———	———
GEO024-3	38.19	120.73	———	———	———

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
GEO025-2	———	1,742.37	———	———	———
GEO026-2	322.11	40.97	———	———	———
GEO026-3	60.75	172.54	———	———	———
GEO027-2	199.83	———	———	———	———
GEO027-3	58.19	———	———	———	———
GEO030-2	———	105.91	———	———	———
GEO030-3	373.93	———	———	———	———
GEO034-3	446.13	———	———	———	———
GEO035-2	2.75	17.41	8.49	———	10.96
GEO035-3	3.19	19.30	10.54	———	13.36
GEO036-2	———	———	95.99	———	———
GEO036-3	162.12	140.93	———	210.89	———
GEO038-2	9.49	28.04	23.85	———	———
GEO038-3	22.32	30.75	46.39	———	———
GEO039-2	———	———	775.34	———	———
GEO039-3	109.26	223.74	———	———	———
GEO041-3	165.31	43.13	———	95.01	———
GEO044-3	295.78	———	———	———	———
GEO052-3	570.11	———	———	———	———
GEO053-3	430.10	———	———	———	———
GEO054-2	10.31	———	191.49	———	———
GEO054-3	3.36	74.34	230.86	———	———
GEO055-2	10.25	———	———	———	———
GEO055-3	27.69	———	———	———	———
GEO056-2	726.64	1,148.80	56.84	———	494.29
GEO056-3	36.66	———	68.84	———	554.99
GEO057-2	10.78	22.45	13.70	———	20.16
GEO057-3	3.48	16.22	17.35	———	20.00
GEO058-2	———	207.13	———	———	———
GEO058-3	———	29.41	———	———	———

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
GEO059-3	73.97	—	—	—	—
GEO064-3	428.77	99.79	135.25	153.22	214.72
GEO065-3	417.90	108.84	153.75	156.27	219.28
GEO066-3	416.81	113.57	168.51	161.49	223.93
GEO067-2	—	38.30	80.15	—	—
GEO067-3	452.37	—	—	—	—
GEO079-1	0.42	0.31	0.30	0.29	0.28
GRP001-1	32.91	6.57	—	—	—
GRP001-5	2.01	1.74	1.77	—	—
GRP003-1	0.52	6.48	—	—	—
GRP003-2	1.57	15.22	—	—	—
GRP004-1	0.49	11.74	—	—	—
GRP004-2	1.57	24.18	—	—	—
GRP005-1	0.39	6.93	0.50	—	0.78
GRP006-1	1.74	6.97	0.59	3.91	1.25
GRP007-1	0.77	1.92	2.08	9.73	25.37
GRP008-1	7.74	542.28	—	—	—
GRP009-1	6.14	13.49	—	—	—
GRP010-1	5.23	9.42	—	—	—
GRP012-1	5.79	22.33	—	—	—
GRP012-2	929.25	29.98	—	—	—
GRP013-1	42.66	90.89	—	—	—
GRP017-1	6.63	5.14	—	—	—
GRP018-1	0.73	4.72	1.90	—	38.95
GRP019-1	0.70	5.41	—	—	39.83
GRP020-1	0.66	67.99	—	4.45	10.61
GRP021-1	0.67	64.79	—	4.45	10.05
GRP022-1	4.25	20.47	—	—	—
GRP022-2	660.26	—	—	—	—
GRP023-1	0.64	1.60	1.68	7.39	43.25

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
GRP023-2	2.25	1.88	60.40	82.73	2.25
GRP025-1	19.33	—	—	—	—
GRP025-2	29.70	—	—	—	—
GRP025-3	19.12	—	—	—	—
GRP025-4	33.03	—	—	—	—
GRP026-1	31.86	—	—	—	—
GRP026-2	41.50	—	—	—	—
GRP026-3	31.83	—	—	—	—
GRP026-4	43.61	—	—	—	—
GRP027-1	1.86	18.07	—	—	—
GRP027-2	1.71	17.31	—	—	—
GRP028-1	0.21	0.29	1.04	161.68	1.11
GRP028-2	0.80	4.05	—	43.99	—
GRP028-3	0.31	1.92	—	330.42	—
GRP029-1	2.73	24.14	—	—	—
GRP029-2	2.56	22.22	—	—	—
GRP030-1	2.65	23.22	—	—	—
GRP031-1	2.60	28.65	—	—	—
GRP031-2	1.37	5.05	—	—	—
GRP032-3	0.82	16.32	—	—	—
GRP033-3	6.42	483.81	—	—	—
GRP033-4	6.51	498.12	—	—	—
GRP034-3	5.93	16.87	—	—	—
GRP034-4	2.22	5.87	0.62	2.98	1.31
GRP035-3	—	121.48	—	—	—
GRP036-3	527.42	17.14	—	—	—
GRP037-3	—	145.91	—	—	—
GRP038-3	0.87	35.50	—	—	—
GRP041-2	0.39	0.45	0.44	0.46	0.48
GRP042-2	1.66	0.78	1.89	4.54	15.92

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
GRP043-2	18.58	1.68	48.53	13.01	40.00
GRP044-2	1.90	2.15	1,061.98	————	————
GRP045-2	1.96	56.80	————	————	————
GRP046-2	367.86	569.21	————	————	————
GRP047-2	366.82	565.87	————	————	————
GRP048-2	————	210.58	————	————	————
HEN001-1	0.73	1.43	1.45	1.94	1.29
HEN001-3	0.51	2.15	0.84	————	1.01
HEN001-5	0.25	0.49	0.48	0.50	0.49
HEN002-1	0.74	1.64	1.66	1.83	1.28
HEN002-2	0.80	1.68	1.74	2.05	1.51
HEN002-3	0.54	1.14	1.15	1.80	1.01
HEN002-4	0.52	1.58	1.23	2.00	1.17
HEN002-5	0.21	0.49	0.49	0.50	0.49
HEN003-1	————	211.68	————	————	————
HEN003-2	————	199.28	————	————	————
HEN003-3	54.13	131.63	————	————	————
HEN003-4	120.78	129.17	————	————	————
HEN003-5	248.20	————	————	————	————
HEN004-4	749.45	————	————	————	————
HEN005-2	————	64.82	————	————	————
HEN005-3	613.40	————	————	————	————
HEN005-6	1,383.34	————	————	————	————
HEN006-4	————	————	452.41	————	————
HEN007-2	155.29	189.44	————	————	————
HEN007-4	1,739.92	134.63	————	————	————
HEN007-6	151.15	210.97	————	————	————
HEN008-1	725.42	53.50	————	————	————
HEN008-2	203.95	506.24	————	————	————
HEN008-3	————	1,223.44	————	————	————

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
HEN008-4	———	———	929.19	———	———
HEN008-6	———	1,270.36	———	———	———
HEN009-2	73.95	20.77	———	———	———
HEN009-4	311.84	37.65	———	———	———
HEN009-5	1,725.47	182.60	———	———	———
HEN010-2	266.33	278.94	———	———	———
HEN010-7	255.89	269.11	———	———	———
HEN012-1	———	199.81	———	———	———
HEN012-3	55.85	133.82	———	———	———
LAT005-1	1,427.52	———	———	———	———
LAT005-2	1,174.88	———	———	———	———
LCL006-1	100.10	———	———	———	———
LCL007-1	0.25	1,141.70	0.28	0.33	0.25
LCL008-1	5.57	1.40	0.87	———	603.15
LCL009-1	70.71	———	———	———	———
LCL010-1	89.40	———	1,127.98	———	———
LCL011-1	523.16	———	———	———	———
LCL013-1	0.36	0.34	0.35	———	14.98
LCL014-1	746.97	———	———	———	———
LCL022-1	70.42	———	2.10	———	———
LCL023-1	69.39	———	———	———	———
LCL025-1	916.38	———	———	———	———
LCL027-1	3.04	58.88	354.04	———	———
LCL033-1	25.66	4.61	88.13	———	———
LCL034-1	990.61	697.70	———	———	———
LCL035-1	3.41	1.45	38.31	———	———
LCL036-1	604.06	326.69	879.11	———	———
LCL039-1	1,568.69	———	———	———	———
LCL041-1	1.77	———	———	———	———
LCL043-1	1.79	176.97	4.81	———	1,118.33

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
LCL044-1	72.50	220.29	———	———	———
LCL045-1	476.27	———	———	———	———
LCL046-1	0.70	1.62	1.28	———	10.93
LCL047-1	1,647.75	———	———	———	———
LCL049-1	1,654.28	———	———	———	———
LCL059-1	1,654.37	———	———	———	———
LCL064-1	1,163.95	———	———	———	———
LCL064-2	234.78	———	———	———	———
LCL065-1	460.58	1,083.93	———	———	———
LCL066-1	164.53	492.68	———	———	———
LCL068-1	0.11	0.28	0.26	0.36	0.27
LCL069-1	0.12	0.28	0.26	0.29	0.28
LCL072-1	200.62	———	———	———	———
LCL075-1	732.47	518.26	———	———	———
LCL076-1	178.30	448.09	515.12	———	———
LCL076-2	0.30	0.45	1.66	9.55	32.83
LCL076-3	38.66	273.65	512.69	———	———
LCL077-1	163.95	444.96	481.69	———	———
LCL077-2	24.12	280.32	496.97	———	———
LCL079-1	4.66	9.39	1,628.06	———	———
LCL081-1	3.17	191.06	93.38	———	———
LCL082-1	22.09	240.07	147.32	———	———
LCL083-1	———	753.76	541.93	———	———
LCL083-2	0.25	0.20	0.22	0.76	0.39
LCL084-2	0.22	0.20	0.22	0.76	0.40
LCL084-3	0.27	0.31	0.32	1.00	0.42
LCL086-1	46.48	21.82	15.84	———	———
LCL087-1	46.42	11.27	57.09	———	———
LCL088-1	46.60	161.98	94.71	———	———
LCL089-1	1,097.58	929.67	———	———	———

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
LCL090-1	292.29	602.74	674.06	———	———
LCL091-1	330.83	668.19	632.12	———	———
LCL092-1	———	815.34	827.37	———	———
LCL094-1	1,211.89	1,664.14	———	———	———
LCL095-1	———	———	1,430.99	———	———
LCL096-1	14.25	———	———	———	———
LCL097-1	33.51	———	———	———	———
LCL098-1	9.43	857.13	775.56	———	———
LCL101-1	14.68	———	———	———	———
LCL102-1	55.63	———	———	———	———
LCL104-1	37.17	———	———	———	———
LCL106-1	4.59	1.75	1.59	———	———
LCL107-1	2.54	———	———	———	———
LCL108-1	50.02	———	8.00	———	———
LCL109-5	———	436.68	———	———	———
LCL110-1	146.03	174.71	289.83	———	———
LCL111-1	102.28	———	———	———	———
LCL112-1	202.85	164.43	299.84	———	———
LCL115-1	———	469.58	666.09	———	———
LCL117-1	0.77	0.55	0.49	———	8.02
LCL118-1	82.67	393.01	424.72	———	———
LCL120-1	37.14	1,651.80	———	———	———
LCL121-1	950.18	———	———	———	———
LCL123-1	368.51	———	———	———	———
LCL126-1	10.96	2.30	2.31	———	———
LCL130-1	6.03	11.48	———	———	———
LCL132-1	61.69	———	———	———	703.43
LCL133-1	213.37	———	———	———	———
LCL143-1	1,731.61	———	———	———	———
LCL169-1	0.37	2.39	0.68	1.01	0.53

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
LCL170-1	0.35	0.64	0.59	1.38	0.52
LCL171-1	0.35	17.49	0.88	7.50	0.55
LCL172-1	0.33	———	1,164.53	———	14.24
LCL173-1	0.39	———	———	———	60.81
LCL174-1	0.87	———	———	———	———
LCL175-1	0.36	0.67	0.60	1.90	0.57
LCL176-1	0.94	1.00	23.43	1.82	3.41
LCL177-1	3.13	1.19	24.02	1.92	3.53
LCL178-1	3.19	1.16	24.58	4.27	3.05
LCL182-1	454.31	———	———	———	———
LCL185-1	0.89	18.23	———	23.98	158.16
LCL186-1	0.86	18.30	———	78.61	72.70
LCL187-1	3.12	19.72	———	81.43	75.41
LCL188-1	3.19	14.88	67.08	13.41	70.79
LCL189-1	3.10	15.05	67.13	34.09	31.95
LCL190-1	0.92	83.36	———	21.55	217.96
LCL192-1	114.08	———	———	558.90	———
LCL193-1	16.48	———	———	———	———
LCL194-1	113.68	———	———	———	———
LCL197-1	3.19	889.17	1,386.55	8.51	———
LCL199-1	180.43	336.18	———	141.67	226.61
LCL200-1	67.39	41.84	———	138.62	15.49
LCL201-1	454.59	———	———	———	———
LCL202-1	1,085.97	688.71	———	477.48	163.72
LCL203-1	461.86	688.92	———	1,556.53	127.14
LCL204-1	458.67	———	———	———	———
LCL205-1	1,090.64	688.23	———	———	218.72
LCL206-1	458.06	690.12	———	1,585.01	107.93
LCL207-1	153.48	699.60	———	———	———
LCL208-1	1,090.40	———	———	———	———

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
LCL211-1	374.61	———	———	———	———
LCL212-1	0.91	6.59	1.10	0.97	0.78
LCL213-1	375.22	———	———	———	———
LCL214-1	375.87	———	———	———	———
LCL216-1	374.67	———	———	———	———
LCL217-1	387.86	———	———	———	———
LCL226-1	0.91	———	———	———	119.39
LDA003-1	156.54	367.51	1,527.18	———	———
LDA007-3	818.94	55.06	295.86	———	———
MSC002-1	60.77	860.04	———	56.33	———
MSC002-2	16.40	131.19	———	39.34	———
MSC003-1	1.82	1.90	1.86	0.73	0.79
MSC004-1	46.01	258.77	247.37	1.44	1.51
MSC005-1	81.24	48.66	24.09	0.53	0.52
MSC006-1	42.01	92.04	91.19	89.99	52.17
MSC007-2.002	3.15	2.77	16.01	———	———
MSC007-2.003	25.12	12.99	95.87	———	———
MSC007-2.004	90.34	68.35	827.64	———	———
MSC007-2.005	291.77	917.41	———	———	———
MSC007-2.006	668.28	———	———	———	———
MSC007-2.007	1,772.50	———	———	———	———
MSC008-1.002	———	1,564.35	1,589.37	860.86	831.97
NUM001-1	1,592.35	———	———	———	———
NUM002-1	66.67	———	———	———	———
NUM003-1	1,575.98	———	———	———	———
NUM004-1	66.37	———	———	———	———
NUM014-1	0.58	0.47	0.64	1.97	0.93
NUM015-1	5.06	1.65	2.41	1,308.48	286.01
NUM016-1	2.67	1.51	1.36	———	9.08
NUM016-2	1.36	0.82	0.80	5.53	1.66

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
NUM019-1	0.41	0.56	1.64	0.57	1.18
NUM020-1	1.58	11.55	344.75	—	—
NUM021-1	9.20	158.28	—	—	—
NUM022-1	1.30	1.10	1.15	—	111.34
NUM023-1	0.52	2.61	4.56	—	2.03
NUM024-1	903.47	—	—	—	—
NUM025-1	0.51	1.14	36.47	1.49	442.32
NUM025-2	0.52	1.24	37.69	1.38	461.60
NUM027-1	78.44	—	—	—	—
NUM139-1	316.02	—	—	—	—
NUM228-1	322.87	—	202.94	—	258.11
NUM283-1.002	1.04	1.03	1.81	1.65	0.76
NUM283-1.003	7.43	3.21	5.32	15.68	3.97
NUM283-1.004	64.58	35.88	49.03	—	552.37
NUM284-1.002	0.85	0.60	1.82	1,106.71	1,270.47
NUM284-1.003	1.14	0.69	1.99	1,794.09	1,554.48
NUM284-1.004	1.60	0.79	2.10	—	—
NUM284-1.005	4.05	1.05	2.51	—	—
NUM284-1.006	5.32	1.61	2.73	—	—
NUM284-1.007	10.62	2.60	3.07	—	—
NUM284-1.008	33.11	4.98	5.75	—	—
NUM284-1.009	69.24	11.34	11.83	—	—
NUM284-1.010	212.23	30.75	30.16	—	—
PLA001-1	—	—	—	4.74	5.13
PLA002-1	1.55	1.14	1.14	1.76	1.16
PLA003-1	1.99	8.08	7.54	1.41	1.39
PLA006-1	9.09	408.36	—	5.40	5.33
PLA017-1	174.96	408.36	—	6.21	7.11
PLA020-1	1.10	25.61	—	4.24	6.16
PLA022-1	—	—	—	304.68	240.22

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
PLA022-2	———	———	———	304.92	239.63
PRV001-1	84.86	31.30	31.49	33.27	33.35
PRV002-1	71.69	———	———	———	———
PRV003-1	0.96	781.35	———	581.46	———
PRV005-1	0.92	———	———	———	———
PRV006-1	7.14	———	———	———	———
PRV007-1	13.08	———	———	———	———
PRV009-1	5.57	30.77	21.82	0.72	0.71
PUZ001-1	1.39	1.00	1.00	1.37	1.37
PUZ001-2	26.17	1,444.66	———	1,478.44	———
PUZ002-1	8.07	1.08	1.08	0.96	0.97
PUZ003-1	1.54	0.90	0.92	1.08	1.10
PUZ005-1	1,533.35	296.11	693.92	318.72	701.31
PUZ006-1	8.79	6.08	9.48	532.96	36.35
PUZ007-1	16.07	9.49	5.66	84.57	9.18
PUZ008-1	0.79	8.39	5.59	5.92	3.13
PUZ010-1	764.99	1,269.36	1,135.95	———	———
PUZ011-1	0.79	1.03	1.03	1.02	1.01
PUZ012-1	1.92	2.72	2.72	2.76	2.80
PUZ018-1	0.97	4.60	4.56	4.56	4.56
PUZ020-1	1.52	1.62	1.66	9.13	12.21
PUZ022-1	3.72	3.89	3.87	7.00	7.17
PUZ023-1	1.78	1.51	1.66	1.49	1.69
PUZ024-1	0.69	1.83	1.75	1.83	1.75
PUZ025-1	4.97	2.29	2.43	2.32	2.58
PUZ026-1	13.36	17.83	18.78	23.14	28.94
PUZ027-1	3.93	2.82	6.45	2.55	7.64
PUZ028-1	25.19	5.91	5.92	731.86	963.79
PUZ028-2	3.18	2.32	2.35	3.17	3.19
PUZ029-1	3.46	1.52	1.53	1.86	1.91

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
PUZ030-1	11.60	12.72	12.87	14.55	14.62
PUZ031-1	13.25	7.18	8.94	———	———
PUZ032-1	1.49	1.55	1.66	14.40	14.12
RNG001-3	315.31	———	———	———	———
RNG002-1	10.02	44.29	———	———	———
RNG003-1	10.13	42.88	———	———	———
RNG005-1	———	1,226.42	———	———	———
RNG005-2	1.30	73.32	———	———	———
RNG006-1	13.05	———	———	———	———
RNG006-2	170.36	———	———	———	———
RNG010-1	17.51	95.45	516.60	12.26	6.58
RNG010-2	0.97	4.68	4.65	4.71	4.74
RNG011-5	0.80	6.39	6.43	6.42	6.50
RNG027-2	0.91	4.66	4.51	4.66	4.61
RNG028-2	0.88	4.61	4.60	4.58	4.61
RNG029-2	0.90	4.63	4.60	4.62	4.60
RNG037-1	———	270.84	———	———	———
RNG037-2	9.67	71.91	———	———	———
RNG038-1	659.67	———	———	———	———
RNG038-2	———	1,105.38	34.56	———	6.33
RNG039-2	1,779.73	———	———	———	———
RNG040-1	1.76	———	———	———	———
RNG040-2	18.27	———	———	———	———
RNG041-1	354.37	———	———	———	———
ROB012-1	0.36	1.56	1.48	1.55	1.57
ROB012-2	0.40	1.70	1.68	1.76	1.71
ROB021-1	171.73	119.73	———	———	———
SET001-1	0.56	0.85	1.54	1.60	0.59
SET002-1	1,502.50	324.08	664.34	523.69	628.97
SET003-1	3.66	280.65	5.59	325.47	8.47

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SET004-1	3.77	1,653.78	5.61	———	8.47
SET006-1	3.26	272.60	6.36	279.14	9.97
SET008-1	233.10	———	———	———	———
SET016-1	52.83	24.03	37.21	———	———
SET016-7	16.85	———	———	———	———
SET018-7	16.98	———	———	———	———
SET024-6	405.88	———	———	———	———
SET024-7	957.89	———	———	———	———
SET025-6	8.87	———	———	———	———
SET025-7	13.76	———	———	———	———
SET043-5	0.19	0.08	0.09	0.08	0.10
SET044-5	9.57	2.95	2.92	2.92	2.98
SET045-5	0.40	0.57	0.16	1.39	1.90
SET046-5	0.49	0.31	0.31	0.32	0.30
SET047-5	2.48	———	———	———	———
SET052-6	400.67	———	———	———	———
SET053-6	399.82	———	———	———	———
SET054-6	9.09	52.20	57.29	51.02	51.02
SET054-7	9.30	55.85	61.29	54.44	54.56
SET055-7	9.33	298.11	———	122.83	———
SET056-6	190.29	———	———	———	———
SET056-7	198.13	———	———	———	———
SET057-6	178.36	———	———	———	———
SET057-7	202.14	———	———	———	———
SET058-6	178.29	———	———	———	———
SET058-7	202.52	———	———	———	———
SET059-6	183.35	———	———	———	———
SET059-7	190.60	———	———	———	———
SET060-6	397.08	———	536.88	———	668.99
SET060-7	497.86	———	———	———	700.98

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SET061-7	625.98	———	———	———	———
SET062-7	9.92	———	———	———	1,583.91
SET063-7	10.24	327.40	———	148.12	———
SET064-7	10.11	———	———	———	———
SET065-6	492.67	626.23	———	311.68	———
SET065-7	772.28	642.54	———	343.54	———
SET072-7	1,024.05	———	———	———	———
SET073-7	884.64	———	———	———	———
SET074-7	883.83	———	———	———	———
SET075-7	868.49	———	———	———	———
SET077-6	9.08	———	———	———	———
SET077-7	11.29	———	———	———	———
SET078-6	386.51	———	———	———	———
SET078-7	11.10	———	———	———	———
SET079-7	280.13	———	———	———	———
SET080-7	11.59	———	123.42	———	104.23
SET081-7	1,025.13	———	———	———	———
SET082-7	1,047.71	———	———	———	———
SET090-7	349.63	———	———	———	———
SET093-6	426.26	———	———	———	———
SET093-7	12.92	———	———	———	———
SET095-7	390.22	———	———	———	———
SET101-7	1,181.67	———	———	———	———
SET102-6	386.35	———	———	———	———
SET117-6	414.45	———	———	———	———
SET117-7	15.41	———	———	———	———
SET118-6	8.88	1,163.08	———	577.73	———
SET118-7	15.91	1,672.34	———	919.43	———
SET158-6	631.22	———	———	———	———
SET203-6	595.12	———	———	———	———

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
SET204-6	585.64	———	———	———	———
SET231-6	153.42	———	———	———	———
SET242-6	592.26	———	———	———	———
SET296-6	3.95	67.84	67.72	67.84	67.84
SYN002-1.000	0.17	0.03	0.04	0.03	0.05
SYN002-1.002	1.52	0.60	0.67	0.60	0.68
SYN002-1.004	12.07	3.14	3.85	3.14	3.83
SYN002-1.006	138.84	9.43	11.98	9.40	12.06
SYN002-1.008	1,192.63	21.20	28.09	21.22	28.22
SYN002-1.010	———	40.45	54.88	40.47	54.92
SYN002-1.012	———	69.25	95.30	69.25	95.36
SYN002-1.016	———	164.08	230.40	164.27	230.42
SYN002-1.020	———	323.77	460.52	324.07	460.57
SYN002-1.030	———	1,145.91	1,658.96	1,146.68	1,662.59
SYN005-1.002	0.10	0.05	0.05	0.05	0.05
SYN005-1.003	0.16	0.08	0.08	0.08	0.07
SYN005-1.004	0.17	0.13	0.11	0.12	0.12
SYN005-1.005	0.23	0.17	0.18	0.18	0.19
SYN005-1.006	0.24	0.26	0.25	0.25	0.25
SYN005-1.007	0.30	0.33	0.34	0.32	0.32
SYN005-1.008	0.34	0.41	0.40	0.39	0.40
SYN005-1.009	0.34	0.51	0.51	0.51	0.52
SYN005-1.010	0.38	0.61	0.61	0.62	0.61
SYN006-1	0.72	0.45	0.46	0.51	0.50
SYN009-1	0.28	0.28	0.28	0.30	0.28
SYN012-1	1,595.79	———	———	———	———
SYN014-2	5.37	2.77	3.44	10.24	10.10
SYN031-1	2.89	0.62	0.56	0.65	0.55
SYN033-1	0.21	0.64	29.22	318.86	1.43
SYN034-1	0.42	0.52	0.31	0.54	0.32

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SYN035-1	0.14	0.11	0.11	0.12	0.12
SYN036-3	15.47	14.20	10.97	14.25	11.02
SYN036-4	———	828.96	100.01	827.27	100.09
SYN037-1	15.64	9.65	8.91	9.61	9.02
SYN037-2	58.55	10.85	20.85	10.28	20.83
SYN038-1	1,580.84	———	———	———	———
SYN048-1	0.05	0.03	0.02	0.02	0.03
SYN049-1	0.06	0.06	0.08	0.05	0.05
SYN050-1	0.16	0.12	0.12	0.13	0.11
SYN051-1	0.36	0.13	0.13	0.12	0.13
SYN052-1	0.24	0.14	0.13	0.13	0.13
SYN053-1	0.33	0.18	0.20	0.18	0.18
SYN054-1	0.73	0.19	0.20	0.29	0.30
SYN055-1	0.88	0.49	0.48	0.57	0.59
SYN056-1	5.10	0.93	0.92	0.98	0.91
SYN057-1	0.67	0.25	0.23	0.28	0.28
SYN058-1	0.38	0.21	0.21	0.25	0.25
SYN059-1	86.27	10.69	11.72	11.40	11.42
SYN060-1	0.43	0.22	0.23	0.22	0.24
SYN061-1	0.39	0.17	0.16	0.16	0.17
SYN062-1	0.62	0.23	0.20	0.25	0.25
SYN063-1	0.27	0.23	0.21	0.22	0.20
SYN064-1	0.08	0.03	0.04	0.04	0.02
SYN065-1	0.28	0.28	0.26	0.29	0.27
SYN066-1	0.55	0.39	0.38	0.40	0.38
SYN068-1	0.87	0.26	0.26	0.27	0.23
SYN069-1	1.62	1.14	1.11	1.47	1.24
SYN070-1	8.98	2.04	2.68	2.36	4.19
SYN071-1	2.54	1.51	1.52	7.35	6.28
SYN072-1	137.94	38.94	30.86	18.97	15.60

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
SYN073-1	0.30	0.41	0.44	6.36	0.23
SYN079-1	0.13	0.12	0.11	5.51	0.25
SYN080-1	0.27	0.50	0.43	8.72	0.32
SYN081-1	0.25	0.23	0.23	0.21	0.22
SYN082-1	2.09	0.79	0.47	1.01	0.40
SYN083-1	0.29	4.28	2.84	—	0.57
SYN084-1	0.29	0.30	0.27	0.29	0.27
SYN088-1.005	0.34	0.36	0.36	0.35	0.35
SYN088-1.010	0.69	1.16	1.15	1.12	1.15
SYN088-1.020	1.43	4.20	4.18	4.18	4.18
SYN095-1.005	18.73	11.52	11.51	26.65	27.30
SYN095-1.010	484.00	153.20	153.11	589.21	602.24
SYN096-1.005	26.02	6.29	6.27	15.21	15.27
SYN096-1.010	173.40	21.96	21.92	58.49	59.44
SYN096-1.020	—	81.17	81.09	246.34	248.40
SYN099-1.005	50.18	33.90	33.93	57.31	58.80
SYN099-1.010	1,395.51	393.52	393.52	1,118.02	1,157.48
SYN100-1.005	73.39	15.18	15.28	26.55	26.99
SYN100-1.010	601.09	49.55	49.47	99.48	101.15
SYN100-1.020	—	172.16	172.37	425.96	431.65
SYN101-1.005	—	—	—	192.39	85.85
SYN101-1.010	—	—	—	—	1,195.05
SYN101-2.005	—	—	—	379.46	206.04
SYN101-3.005	—	—	—	923.10	597.08
SYN102-1.005	—	153.17	986.54	48.01	58.16
SYN102-1.010	—	467.86	—	150.96	175.87
SYN102-1.020	—	1,550.87	—	573.36	606.18
SYN102-2.005	—	—	—	117.95	139.02
SYN102-2.010	—	—	—	307.60	388.04
SYN102-2.020	—	—	—	912.97	1,209.09

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
SYN102-3.005	—	—	—	374.22	413.24
SYN102-3.010	—	—	—	868.91	1,052.27
SYN103-1	8.85	108.79	108.08	110.50	108.16
SYN104-1	9.00	108.76	108.10	108.86	108.09
SYN105-1	30.40	111.03	110.35	111.15	110.36
SYN106-1	30.04	126.22	125.28	111.18	110.60
SYN107-1	30.30	329.51	291.76	113.62	111.96
SYN108-1	30.50	319.75	282.83	118.61	116.67
SYN109-1	139.73	1,695.14	1,604.74	185.22	179.46
SYN110-1	138.73	—	—	210.77	204.06
SYN111-1	137.39	1,790.23	1,739.05	193.05	187.46
SYN112-1	30.54	143.87	280.17	113.70	113.83
SYN113-1	139.94	1,605.46	1,498.18	185.85	181.79
SYN114-1	80.31	243.42	199.56	123.51	120.86
SYN115-1	223.39	—	—	187.53	175.24
SYN116-1	139.09	565.38	562.86	299.76	261.54
SYN117-1	241.36	510.03	506.94	290.33	248.11
SYN118-1	8.97	108.72	108.29	108.77	108.12
SYN119-1	8.77	108.80	108.25	108.71	108.19
SYN120-1	30.36	141.16	277.98	111.21	111.48
SYN121-1	139.97	828.88	785.82	149.86	160.80
SYN122-1	137.48	829.79	789.31	167.16	163.91
SYN123-1	80.02	821.50	770.04	199.59	184.67
SYN124-1	79.86	162.30	132.97	170.25	133.43
SYN125-1	136.65	711.15	567.13	131.94	126.80
SYN126-1	136.42	288.80	241.31	142.91	136.80
SYN127-1	136.87	296.10	248.53	244.18	212.61
SYN128-1	422.67	—	—	285.21	240.80
SYN129-1	423.43	—	—	284.05	242.51
SYN130-1	29.64	—	—	187.36	181.10

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SYN131-1	8.70	111.62	110.71	111.51	110.75
SYN132-1	30.06	———	———	187.04	180.95
SYN133-1	29.77	———	———	187.05	180.85
SYN134-1	80.01	146.41	390.29	124.72	123.78
SYN135-1	90.60	259.10	212.46	128.32	124.99
SYN136-1	91.31	257.66	212.77	127.90	125.21
SYN137-1	238.79	874.72	824.17	210.00	200.92
SYN138-1	238.64	———	———	206.44	199.01
SYN139-1	649.69	———	———	342.38	317.18
SYN140-1	648.84	———	———	318.84	308.08
SYN141-1	237.46	———	———	272.92	273.20
SYN142-1	648.72	———	———	368.25	354.07
SYN143-1	650.48	———	———	368.35	353.32
SYN144-1	1,412.67	744.10	708.73	179.50	173.08
SYN145-1	8.68	111.22	110.52	111.21	110.52
SYN146-1	9.25	115.93	114.92	115.94	114.88
SYN147-1	9.27	115.74	114.68	115.76	114.67
SYN148-1	79.89	———	———	162.67	158.46
SYN149-1	29.52	346.10	285.66	122.26	118.73
SYN150-1	79.82	190.15	145.37	129.01	124.36
SYN151-1	79.29	201.18	159.92	127.44	123.59
SYN152-1	29.83	204.38	149.01	118.10	115.20
SYN153-1	79.24	216.01	174.55	146.06	140.47
SYN154-1	79.41	224.29	182.00	130.18	126.10
SYN155-1	237.46	1,481.36	———	———	———
SYN156-1	1,411.05	1,496.45	———	———	———
SYN157-1	423.26	1,638.26	———	558.48	1,211.98
SYN158-1	398.26	840.71	799.01	556.47	1,226.38
SYN159-1	1,410.32	———	———	———	1,438.68
SYN160-1	400.26	841.06	799.29	613.14	1,210.94

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SYN161-1	237.18	—	—	644.39	1,219.38
SYN162-1	237.11	840.67	799.10	643.51	1,216.86
SYN163-1	1,414.65	—	—	—	1,471.60
SYN164-1	8.84	108.85	108.03	108.91	108.16
SYN165-1	9.46	111.04	110.50	111.11	110.41
SYN166-1	136.64	1,136.98	1,110.60	167.54	155.95
SYN167-1	29.75	196.27	152.88	115.92	113.68
SYN168-1	80.56	278.78	232.82	134.62	129.68
SYN169-1	79.42	281.73	234.92	232.86	202.79
SYN170-1	136.17	508.74	486.30	177.11	167.75
SYN171-1	1,413.30	—	—	407.36	412.27
SYN172-1	8.82	108.94	108.14	108.86	108.44
SYN173-1	29.86	958.42	900.26	128.28	123.76
SYN174-1	29.88	983.75	924.55	128.26	123.71
SYN175-1	29.55	351.50	316.49	116.46	114.84
SYN176-1	136.96	820.98	772.26	311.31	291.25
SYN177-1	135.46	934.42	882.35	187.93	138.23
SYN178-1	236.87	930.66	878.40	932.47	560.65
SYN179-1	238.66	1,330.00	1,283.84	638.92	636.13
SYN180-1	238.30	—	—	435.00	426.03
SYN181-1	136.63	—	—	185.67	206.43
SYN182-1	457.36	293.74	246.47	347.46	149.98
SYN183-1	238.06	298.47	250.91	313.63	215.17
SYN184-1	8.66	108.77	108.17	108.71	108.21
SYN185-1	8.95	109.08	108.27	108.77	108.19
SYN186-1	29.69	502.24	493.86	117.88	115.96
SYN187-1	29.66	499.70	496.04	117.43	115.87
SYN188-1	29.82	496.26	493.97	117.46	116.24
SYN189-1	135.36	503.45	498.68	120.24	118.88
SYN190-1	422.22	—	—	302.49	400.90

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
SYN191-1	237.59	—	—	272.38	234.05
SYN192-1	237.84	833.57	794.78	254.08	314.76
SYN193-1	237.94	833.72	793.02	259.51	303.79
SYN194-1	237.64	516.11	512.15	221.65	205.20
SYN195-1	237.80	517.82	513.92	178.18	168.47
SYN196-1	79.62	159.74	403.21	191.59	186.16
SYN197-1	29.86	129.11	128.66	111.33	110.53
SYN198-1	78.79	160.28	401.59	191.82	185.65
SYN199-1	79.76	160.15	404.05	191.69	186.02
SYN200-1	80.12	160.10	404.08	192.86	185.64
SYN201-1	79.57	355.20	309.52	144.13	130.42
SYN202-1	238.26	572.83	609.70	302.66	281.44
SYN203-1	137.28	—	—	316.18	212.91
SYN204-1	424.08	—	—	300.44	273.32
SYN205-1	424.17	—	—	290.41	263.62
SYN206-1	238.02	—	—	177.54	168.89
SYN207-1	422.23	—	—	204.81	190.39
SYN208-1	137.02	—	—	216.37	182.26
SYN209-1	136.66	509.93	507.15	151.98	146.37
SYN210-1	136.84	512.46	508.88	163.28	158.00
SYN211-1	136.88	513.93	510.34	149.30	144.64
SYN212-1	135.78	516.38	511.07	139.64	136.40
SYN213-1	398.74	834.83	793.89	498.58	355.08
SYN214-1	397.85	835.84	794.29	952.97	349.23
SYN215-1	399.88	834.58	794.88	463.53	342.47
SYN216-1	79.15	820.35	768.52	237.11	195.45
SYN217-1	79.87	821.29	770.32	227.43	179.37
SYN218-1	79.38	826.55	783.35	152.73	165.32
SYN219-1	134.96	824.05	781.39	149.63	160.02
SYN220-1	80.10	165.19	148.04	151.16	137.17

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SYN221-1	79.92	173.25	160.45	168.73	175.87
SYN222-1	79.17	173.41	164.80	153.60	138.35
SYN223-1	79.50	176.18	138.81	158.95	131.23
SYN224-1	79.47	162.39	132.97	169.92	133.36
SYN225-1	79.16	702.76	559.78	131.98	126.76
SYN226-1	79.89	165.03	147.73	151.20	137.15
SYN227-1	136.55	706.13	562.91	128.80	123.69
SYN228-1	136.03	693.31	559.98	127.42	122.44
SYN229-1	80.09	170.71	160.30	166.97	155.73
SYN230-1	80.48	192.87	175.26	172.58	158.68
SYN231-1	79.40	176.56	165.14	157.73	161.48
SYN232-1	79.91	197.74	180.91	161.55	146.92
SYN233-1	79.74	175.16	138.91	160.16	132.78
SYN234-1	79.84	716.19	568.99	135.70	129.98
SYN235-1	136.16	737.06	579.63	132.59	127.50
SYN236-1	79.96	174.07	138.18	158.97	131.55
SYN237-1	8.81	111.32	110.66	111.17	110.35
SYN238-1	30.18	312.27	263.08	118.51	116.22
SYN239-1	9.06	112.88	111.75	112.13	112.10
SYN240-1	8.97	111.31	110.64	111.19	110.54
SYN241-1	9.12	111.20	110.87	111.35	110.65
SYN242-1	9.01	108.89	108.23	108.82	108.20
SYN243-1	30.03	659.39	656.51	120.44	119.44
SYN244-1	8.96	111.17	110.70	111.19	110.94
SYN245-1	8.94	111.48	110.70	111.37	110.70
SYN246-1	29.95	530.01	462.30	118.38	116.23
SYN247-1	29.94	314.85	262.75	113.70	114.27
SYN248-1	79.37	587.14	545.63	211.41	138.32
SYN249-1	79.77	588.32	545.78	177.19	137.93
SYN250-1	136.90	1,152.20	1,124.47	397.53	245.72

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	CLIN-2	CLIN-2	CLIN-2	CLIN-2
SYN251-1	30.23	661.95	659.40	120.46	119.38
SYN252-1	649.12	———	———	360.12	332.68
SYN253-1	646.63	———	———	330.92	318.77
SYN254-1	649.27	———	———	336.85	318.79
SYN255-1	30.04	207.81	147.03	124.58	121.73
SYN256-1	79.45	227.76	173.30	139.55	134.21
SYN257-1	8.81	108.74	108.11	108.76	108.42
SYN258-1	29.93	196.40	153.04	115.95	113.65
SYN259-1	30.02	196.16	152.76	115.76	113.67
SYN260-1	30.38	196.15	152.88	115.83	113.59
SYN261-1	29.84	187.46	143.00	113.54	111.26
SYN262-1	79.42	668.77	667.34	124.24	122.65
SYN263-1	137.66	1,228.29	1,195.04	138.28	136.69
SYN264-1	79.69	678.85	676.94	124.68	123.35
SYN265-1	79.80	273.53	227.26	127.88	124.92
SYN266-1	237.46	1,408.34	1,360.18	250.95	247.02
SYN267-1	136.09	478.97	443.05	152.42	147.57
SYN268-1	136.20	478.52	443.61	133.77	130.86
SYN269-1	1,412.95	———	———	377.82	437.53
SYN270-1	422.95	———	———	276.87	237.90
SYN271-1	1,413.37	———	———	407.97	412.37
SYN272-1	423.35	1,621.17	1,508.39	293.48	373.16
SYN273-1	422.84	———	———	192.48	189.06
SYN274-1	8.82	108.89	108.01	108.86	108.09
SYN275-1	8.69	111.10	110.43	111.06	110.25
SYN276-1	8.78	108.66	107.98	108.76	108.01
SYN277-1	9.82	111.25	110.37	111.25	110.57
SYN278-1	29.68	235.57	193.25	118.48	115.99
SYN279-1	30.14	963.96	905.45	132.58	127.98
SYN280-1	9.97	111.17	110.50	111.26	110.54

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems (continued)

Problem	Forward Support			Backward Support	
	Max		Sum	Max	Sum
	CLIN	**CLIN-2**	**CLIN-2**	**CLIN-2**	**CLIN-2**
SYN281-1	29.99	229.10	187.57	115.77	113.58
SYN282-1	9.76	111.22	110.53	111.06	110.63
SYN283-1	30.26	962.75	904.36	128.12	123.67
SYN284-1	30.24	969.72	908.75	128.23	123.64
SYN285-1	79.67	1,577.42	1,474.65	212.54	149.92
SYN286-1	30.35	969.68	914.96	128.32	123.65
SYN287-1	9.95	111.16	110.61	111.11	110.52
SYN288-1	10.10	108.59	108.07	108.63	108.13
SYN289-1	30.16	972.50	914.22	128.03	123.63
SYN290-1	29.99	229.29	188.43	115.98	113.72
SYN291-1	9.80	111.36	110.46	111.35	110.44
SYN292-1	29.69	312.98	263.50	113.52	111.16
SYN293-1	80.14	771.64	738.07	614.30	134.37
SYN294-1	80.27	849.24	814.93	222.92	150.48
SYN295-1	9.75	108.66	108.06	108.68	108.09
SYN296-1	30.37	982.88	923.62	128.07	123.60
SYN297-1	11.62	108.71	108.14	108.72	108.04
SYN298-1	238.52	———	———	177.32	168.78
SYN299-1	238.18	———	———	177.19	168.63
SYN300-1	237.50	———	———	177.12	168.57
SYN301-1	136.77	501.91	499.66	143.17	139.82
TOP002-2	0.13	0.08	0.07	0.07	0.08
TOP003-2	———	———	———	0.59	0.58
TOP004-1	3.60	5.55	5.58	5.59	5.57
TOP004-2	0.61	1.16	1.18	1.16	1.17

Table B.2: CLIN and CLIN-2 Results on TPTP V1.1.3 Non-Propositional Problems

Bibliography

[1] Geoffrey D. Alexander and David A. Plaisted. Proving equality theorems with hyperlinking. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 706–710, Saratoga Springs, New York, 15–18 June 1992. Springer-Verlag. System abstract.

[2] Peter B. Andrews. Theorem proving via general matings. *J. ACM*, 28(2):193–214, April 1981.

[3] J. Avenhaus and K. Madlener. Term rewriting and equational reasoning. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Sourcebook*, volume 6 of *Studies in Computer Science and Artificial Intelligence*, pages 1–43. Elsevier Science Publishers, Amsterdam, 1990.

[4] Leo Bachmair, Nachum Dershowitz, and Jieh Hsiang. Orderings for equational proofs. In *Proceedings of the Symposium on Logic in Computer Science*, pages 346–357, Cambridge, Massachusetts, 16–18 June 1986.

[5] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures 2: Rewriting Techniques*, pages 1–30. Academic Press, New York, 1989.

[6] Leo Bachmair, Harold Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation and superposition. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 462–476, Saratoga Springs, New York, 15–18 June 1992. Springer-Verlag.

[7] Leo Bachmair and David A. Plaisted. Termination orderings for associative-commutative rewriting systems. *J. Symbolic Computation*, 1:329–349, 1985.

[8] Jon Barwise, editor. *Handbook of Mathematical Logic*, volume 90 of *Studies of Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, 1977.

[9] George Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[10] W. W. Bledsoe and D. W. Loveland, editors. *Automated Theorem Proving: After 25 Years*, volume 29 of *Contemporary Mathematics*. American Mathematical Society, Providence, Rhode Island, 1984.

[11] George S. Boolos and Richard C. Jeffrey. *Computability and Logic, Third Edition*. Cambridge University Press, Cambridge, 1989.

[12] Alexandre Boudet. Unification in a combination of equational theories: an efficient algorithm. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 292–307, Kaiserslautern, West Germany, 24–27 July 1990. Springer-Verlag.

[13] Alexandre Boudet. Unification in order-sorted algebras with overloading. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 193–207, Saratoga Springs, New York, 15–18 June 1992. Springer-Verlag.

[14] D. Brand. Proving theorems with the modification method. *SIAM J. Comput.*, 4(4):412–430, December 1975.

[15] Alan Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, New York, 1983.

[16] Ritu Chadha. *Applications of Unskolemization*. PhD Dissertation, University of North Carolina at Chapel Hill, 1991. Available from the Department of Computer Science as technical report TR91-027.

[17] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[18] Heng Chu. *Semantically Guided First-Order Theorem Proving with Hyper-Linking*. PhD Dissertation, University of North Carolina at Chapel Hill, 1994. Available from the Department of Computer Science as technical report TR94-051.

[19] Heng Chu and David A. Plaisted. Semantically guided first-order theorem proving using hyper-linking. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 192–206, Nancy, France, 26 June –1 July 1994. Springer-Verlag.

[20] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog, Second Edition*. Springer-Verlag, Berlin, 1984.

[21] Paul M. Cohn. *Universal Algebra*, volume 6 of *Mathematics and Its Applications*. D. Reidel Publishing, Dordrecht, Holland, 1981.

[22] Hubert Comon. Solving inequations in term algebras. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 62–69, Philadelphia, Pennsylvania, 4–7 June 1990.

[23] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3–5 May 1971.

[24] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–205, 1960.

[25] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Comput. Sci.*, 17:279–301, 1982.

[26] Nachum Dershowitz. Computing with rewrite systems. *Information and Control*, 65:122–157, 1985.

[27] Nachum Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3:69–116, 1987.

[28] Nachum Dershowitz. Completion and its applications. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures 2: Rewriting Techniques*, pages 31–85. Academic Press, New York, 1989.

[29] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewriting systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier Science Publishers, Amsterdam, 1990.

[30] Daniel J. Dougherty and Patricia Johann. An improved general E-unification method. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 261–275, Kaiserslautern, West Germany, 24–27 July 1990. Springer-Verlag.

[31] David A. Duffy. *Principles of Automated Theorem Proving.* John Wiley and Sons, Chichester, England, 1991.

[32] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic, Second Edition.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1994. Translated from *Einführung in die mathematische Logik*, published by Wissenschaftliche Buchgesellschaft, Darmstadt, by Ann S. Ferebee.

[33] Yu. L. Ershov and E. A. Palyutin. *Mathematical Logic.* Mir Publishers, Moscow, 1984.

[34] François Fages. Associative-commutative unification. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 194–208, Napa, California, 14–16 May 1984. Springer-Verlag.

[35] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, 1990.

[36] Albrecht Fortenbacher. An algebraic approach to unification under associativity and commutativity. In Jean-Pierre Jouannaud, editor, *Proceedings of the 1st International Conference on Rewriting Techniques and Applications*, volume 202 of *Lecture Notes in Computer Science*, pages 381–397, Dijon, France, 20–22 May 1985. Springer-Verlag.

[37] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row, New York, 1986.

[38] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[39] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM J. Res. Dev.*, 4(1):28–35, January 1960.

[40] Kurt Gödel. *Über die Vollständigkeit des Logikkalküls*. Doctoral Dissertation, University of Vienna, 1929. English translation in Gödel[43, pages 60-101].

[41] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930. English translation in Gödel[43, pages 103-123].

[42] Kurt Gödel. Über formal unentscheidbare Sätze der *Principia mathematica* und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. English translation in Gödel[43, pages 145-195].

[43] Kurt Gödel. *Kurt Gödel: Collected Works*, volume I. Oxford University Press, New York, 1986. Edited by Solomon Feferman.

[44] David Gries. *The Science of Programming*. Spring-Verlag, New York, 1981.

[45] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. Thesis, University of Paris, 1930. English translation in Herbrand[46].

[46] Jacques Herbrand. *Jacques Herbrand: Logical Writings*. Harvard University Press, Cambridge, Massachusetts, 1971. Edited by Warren D. Goldfarb.

[47] David Hilbert and Wilhelm Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, New York, 1950. English translation of the second German edition of *Grundzüge der Theoretischen Logik* published in 1938. Translation is by Lewis H. Hammond, George G. Leckie, and F. Steinhardt. Edited and with notes by Robert E. Luce.

[48] Jieh Hsiang. *Topics in Automated Theorem Proving and Program Generation*. PhD Dissertation, University of Illinois at Urbana-Champaign, 1983. Available from University Microfilms International, Ann Arbor, Michigan.

[49] Jieh Hsiang and Jalel Mzali. *SbREVE User's Guide*, July 1988.

[50] Jieh Hsiang and Michael Rusinowitch. On word problems in equational theories. In Thomas Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71, Karlsruhe, Federal Republic of Germany, 13–17 July 1987. Springer-Verlag.

[51] G. Huet. A complete proof of correctness of the Knuth and Bendix completion algorithm. *J. Comput. Syst. Sci.*, 23(1):11–21, August 1981.

[52] IBM Corporation, North York, Ontario. *C Set ++ for AIX/6000: Collection Class Library Reference, SC09-1602-00*, December 1993.

[53] Jean-Pierre Jouannaud and Mitsuhiro Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In J. Leech Albert, B. Monien, and Rodríguez M. Artalejo, editors, *Proceedings of the 18th International Colloquium on Automata, Languages, and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468, Madrid, Spain, 8–12 July 1991. Springer-Verlag.

[54] Donald Kalish, Richard Montague, and Gary Mar. *Logic: Techniques of Formal Reasoning, Second Edition.* Harcourt Brace Jovanovich, Publishers, San Diego, 1980.

[55] S. Kamin and J.-J. Lévy. Two generalizations of the recursive path orderings. Unpublished notes, Department of Computer Science, University of Illinois, Urbana, Illinois, 1980.

[56] Deepak Kapur and Hantao Zhang. Proving equivalence of different axiomizations of free groups. *J. Automated Reasoning*, 4:331–352, 1988.

[57] Deepak Kapur and Hantao Zhang. *RRL: Rewrite Rule Laboratory User's Manual*, May 1989.

[58] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In John Leech, editor, *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, Oxford, 1970.

[59] Robert Kowalski. A proof procedure using connection graphs. *J. ACM*, 22(4):572–595, October 1975.

[60] D. Lankford. Canonical algebraic simplification of computational logic. Technical Report Technical Report Memo ATP-25, Automatic Theorem Proving Project, University of Texas, Austin, Texas, 1975.

[61] D. Lankford. On proving term rewriting systems are noetherian. Technical Report Technical Report Memo MTP-3, Mathematics Department, Louisiana Tech University, Ruston, Louisiana, 1979.

[62] Shie-Jue Lee. *CLIN: An Automated Reasoning System Using Clause Linking.* PhD Dissertation, University of North Carolina at Chapel Hill, 1990. Available from the Department of Computer Science as technical report TR90-029.

[63] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyper-linking strategy. *J. Automated Reasoning*, 9:25–42, 1992.

[64] J. W. Lloyd. *Foundations of Logic Programming, Second, Extended Edition.* Springer-Verlag, Berlin, 1987.

[65] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification, Second Edition.* John Wiley & Sons and B. G. Teubner, Chichester, 1987.

[66] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland Publishing, Amsterdam, 1978.

[67] E. L. Lusk and R. A. Overbeek. Reasoning about equality. *J. Automated Reasoning*, 1:209–228, 1985.

[68] Ursula Martin and Tobias Nipkow. Ordered rewriting and confluence. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 366–380, Kaiserslautern, West Germany, 24–27 July 1990. Springer-Verlag.

[69] John D. McCharen, Ross A. Overbeek, and Lawrence A. Wos. Problems and experiments for and with automated theorem-proving programs. *IEEE Trans. Comput.*, C-25(8):773–782, August 1976.

[70] William W. McCune. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory, Argonne, Illinois, January 1994.

[71] Elliott Mendelson. *Introduction to Mathematical Logic, Third Edition*. Wadsworth and Brooks, Monterey, California, 1987.

[72] Ernest Nagel and James R. Newman. *Gödel's Proof*. New York University Press, 1958.

[73] Paliath Narendran and Friedrich Otto. Some results on equational unification. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 276–291, Kaiserslautern, West Germany, 24–27 July 1990. Springer-Verlag.

[74] Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering constrained clauses. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 477–491, Saratoga Springs, New York, 15–18 June 1992. Springer-Verlag.

[75] E. Paul. On solving the equality problem in theories defined by Horn clauses. *Theoretical Comput. Sci.*, 44:127–153, 1986.

[76] Gerald E. Peterson. Complete sets of reductions with constraints. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 381–395, Kaiserslautern, West Germany, 24–27 July 1990. Springer-Verlag.

[77] Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, April 1981.

[78] David A. Plaisted. An efficient relevance criterion for mechanical theorem proving. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence*, pages 79–83, Stanford University, 18–21 August 1980.

[79] David A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:122–157, 1985.

[80] David A. Plaisted. Non-Horn clause logic programming without contrapositives. *J. Automated Reasoning*, 4:287–325, 1988.

[81] David A. Plaisted. Mechanical theorem proving. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Sourcebook*, volume 6 of *Studies in Computer Science and Artificial Intelligence*, pages 269–320. Elsevier Science Publishers, Amsterdam, 1990.

[82] David A. Plaisted. Equational reasoning and term rewriting systems. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, New York, 1993.

[83] David A. Plaisted. The use of semantics in instance-based proof procedures. Unpublished notes, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 1994.

[84] David A. Plaisted, Geoffrey D. Alexander, Heng Chu, and Shie-Jue Lee. Conditional term rewriting and first-order theorem proving. In *Proceedings of the Third International Workshop on Conditional Term-Rewriting Systems*, Pont-à-Mousson, 8-10 July 1992. Invited talk.

[85] David A. Plaisted and Steve Greenbaum. A structure-preserving clause form translation. *J. Symbolic Computation*, 2:293–304, 1986.

[86] David A. Plaisted and Shie-Jue Lee. Inference by clause linking. Technical Report TR90-022, Department of Computer Science, University of North Carolina at Chapel Hill, May 1990.

[87] David A. Plaisted and Richard Potter. Term rewriting: Some experimental results. *J. Symbolic Computation*, 11:149–180, 1991.

[88] J. A. Robinson. A machine-oriented logic based on the resolution method. *J. ACM*, 12(1):23–41, January 1965.

[89] Jörg H. Siekmann. Universal unification. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 1–42, Napa, California, 14–16 May 1984. Springer-Verlag.

[90] Jörg H. Siekmann. An introduction to unification theory. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Sourcebook*, volume 6 of *Studies in Computer Science and Artificial Intelligence*, pages 369–424. Elsevier Science Publishers, Amsterdam, 1990.

[91] Mark E. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(2):423–434, April 1981.

[92] Abram Aronovich Stolyar. *Introduction to Elementary Mathematical Logic*. The MIT Press, Cambridge, Massachusetts, 1970. English translation of *Elementarnoye vvedeniye v matematicheskuyu logiku* published in 1965. Translation is by Scripta technica, Inc. Edited by Elliott Mendelson.

[93] Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 252–266, Nancy, France, 26 June –1 July 1994. Springer-Verlag.

[94] Christian B. Suttner and Geoff Sutcliffe. The TPTP problem library (TPTP v1.1.1). Technical Report AR-94-03, Institut für Informatik, Technische Universität München, Munich, Germany, 7 July 1994. Also available as technical report 93/11, Department of Computer Science, James Cook University, Townsville, Australia.

[95] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 12 November 1936.

[96] Allen Van Gelder and Yumi K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1995.

[97] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[98] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, October 1965.