Parallel Semantic Tree Theorem Proving with Resolutions

Choon Kyu Kim

School of Computer Science
McGill University, Montreal

July 2004

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Your file Votre référence
ISBN: 0-494-06310-6
Our file Notre référence
ISBN: 0-494-06310-6

Abstract

Semantic trees have often been used as theoretical tools for showing the unsatisfiability of clauses in first-order predicate logic. Their practicality has been overshadowed, however, by other strategies though considerable effort has been made to improve semantic tree theorem provers over the last decade.

In this thesis, we propose building a parallel system through the integration of semantic trees with resolution-refutation. The proposal comes from the observations that the appropriate strategy for one class of theorems is often very different from that for another class and many semantic trees tend to be linear. In the linear semantic tree, one of the two branches from each node leads to a failure node. Such linearity is attractive because we can focus our efforts on closing the remaining branch. Unfortunately, the strategy of building a closed linear semantic tree is incomplete. To help to achieve closure, we introduce the use of unit clauses derived from resolutions when necessary, leading to a strategy that combines the construction of semantic trees with resolution-refutation.

The parallel semantic tree theorem prover, called PrHERBY, utilizes dedicated resolutions in scalable manner and strategically selects atoms to construct semantic trees. In addition, a parallel grounding scheme allows each system to have its own instance of generated atoms, thereby increasing the possibility of success. The PrHERBY system presented performs significantly better and generally finds proof using fewer atoms than the semantic tree prover, HERBY and its parallel version, PHERBY.

Résumé

Les arbres sémantiques ont souvent servi comme des outils théoriques pour démontrer la non-satisfiabilité des clauses dans la logique de premier ordre de prédicat. Leur caractère pratique a, cependant, été éclipsé par d'autres stratégies d'épreuve malgré l'effort qui fut investi dans le but d'améliorer les systèmes basés sur les arbres sémantiques pendant la dernière décénie.

Dans cette thèse, on propose la construction d'un système parallèle par l'intégration des arbres sémantiques avec la résolution-réfutation. Cette idée émerge du fait de remarquer que la stratégie convenable pour une classe de théorèmes est souvent différente de celle qui convient à une autre classe et plusieurs des arbres sémantiques sont linéaires. Un arbre sémantique linéaire est un arbre sémantique dans lequel une des deux branches de chaque nœud mène à un nœud d'échec. Une telle linéarité est intéressante parce qu'elle nous permet de concentrer nos efforts sur la fermeture de la branche restante. Malheureusement, la stratégie de construire un arbre sémantique linéaire est incomplète. Pour réaliser la fermeture, nous introduisons, là où c'est nécessaire, des clauses à unité qui sont dérivées de résolutions. Ceci mène à une stratégie qui combine la construction des arbres sémantiques avec la résolution-réfutation.

Le système parallèle d'aide à la preuve basé sur l'arbre sémantique, appelé Pr-HERBY utilise stratégiquement les atomes avec l'aide des résolutions spécifiques d'une manière qui permet l'échelonnage dans le but de construire les arbres sémantiques. De plus, un arrangement au sol parallèle permet à chaque système

d'avoir sa propre instance des atomes produits, ce qui augmente la possibilité de succés. Le système présenté, PrHERBY, performe d'une façon qui est significativement meilleure et trouve généralement la preuve en utilisant moins d'atomes que HERBY, le système d'aide à la preuve basé seulement sur l'arbre sémantique et sa version parallèle PHERBY.

Acknowledgments

First of all, I would like to thank my supervisor, Professor Monty Newborn, for his guidance and encouragement throughout the course of this research. His meticulous reading and insistence on clarity have been instrumental in writing and clarifying about some of the issues described in this thesis. This thesis could never have been done without him.

I appreciate the members of my thesis committee, Tim Merrett, Brigitte Pientka, Doina Precup, and Peter Caines from ECE department for serving on my committee and giving valuable criticism and support. It was an intense but interesting experience I can enjoy and remember for a long time. In particular, I would like to thank Professor J.D. Horton, the external examiner. His criticism and suggestions upgraded my thesis work considerably. His comments, diligence about reading and efforts to understand what I wrote one by one were certainly more than the role of the external examiner.

I wish to express my gratitude to Korea Telecom for the financial support. I specially wish to thank Dr. Youngkyung Lee for giving me the opportunity to study abroad and encouraging my family and me. I am grateful to my colleagues and company for allowing me to accomplish this work. Special thanks go to Dr. Sanghong Lee who is in charge of the research center.

I would like to thank my uncle for his continuous encouragements. My cousin, Jina, deserves special thanks for reviewing my early work.

A former colleague, Sergei Savchenko, helped me begin the writing with LaTeX. The system staffs at SOCS provided assistance for the experiments whenever needed.

They deserve special thanks. I also would like to thank my colleague, Matthew Suderman, for reading the draft and giving me valuable comments. I appreciate Ms. Susan Rose for reviewing my thesis and telling me how to present my ideas more clearly. I also want to thank Danielle Azar who corrected my résumé. She was one of the graduate students who had experienced the difficult times together in McGILL. My thanks go to my brilliant friend, Professor DongGook Park who encouraged me to finish this thesis.

My school days would not have been successful without the support of my family. I thank my wife, Young Ran, for her love and support. She has cheerily endured years of bluntness from me and shared all the frustration and joy in the exotic environment. My cheerleaders, SungJu and HyunJu, have been my fountains of joy and peace. Finally, I remember that my parents are praying for me and giving me their support. I would like to thank my brothers and mother. I dedicate this thesis to my father who gave me special affection during his lifetime, and who still lives in my mind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automated theorem proving is a subfield in artificial intelligence that is mainly concerned with mechanized reasoning. Sound reasoning shows that conclusions logically follow from facts. Such an argument is called a proof. Although the formulas expressed in first-order logic are generally undecidable, there are proof procedures that can verify that a formula is valid, if indeed it is valid. The fundamental theorem of Herbrand is basis for most proof procedures. The theorem states that we can prove the unsatisfiability of formulas by considering the interpretations over the Herbrand universe instead of all interpretations over all domains. The semantic tree is a systematic way of organizing the interpretations over the Herbrand universe. A set S of clauses is unsatisfiable if and only if there is a finite, closed, semantic tree of S [CL73].

Besides the theoretical foundation of confirming the unsatisfiability of sets of clauses, constructing a semantic tree for determining the unsatisfiability of a set of clauses has been overshadowed by other strategies such as Robinson's *resolution principle*, Loveland's *model elimination* and Kowalski's *connection graph method.*

Considerable effort has been made to improve semantic tree theorem provers over the last decade. Almulla investigated the practicality of a semantic tree theorem prover in his thesis [Alm95]. His prover solved 29 of the 84 Stickel test set problems by simply constructing canonical semantic trees. It solved a total of 47 theorems after some refinement [AN96]. HERBY, the same type of theorem prover designed by Newborn, can now solve all but five of them while THEO, Newborn's resolution-refutation theorem prover, can solve all of them [New01].

It is surprising that semantic tree prover could do well on the Stickel test set. Furthermore, there are many researches available regarding the very similar strategy such as tableau methods. However, the semantic tree theorem prover is still not competitive with resolution-based provers. The motivation of this research is to extend the Almulla's serial work to the parallel case, actually improving the scalability and therefore the performance.

Proving theorems is difficult because the search space of the most interesting theorems is enormous and the direction for the search unclear. Which rules to apply, and in what order, is not necessarily obvious. It is a well-established observation that different strategies work for different sets of theorems. In order to build more powerful systems, besides struggling to improve the semantic tree method, integrating it with different strategies offers interesting prospects. We believe that the semantic tree method could play an important role in automated theorem proving due to its inherent semantic-oriented nature and its correspondence with the resolution-refutation method.

In this thesis, we first investigate linear properties of semantic trees. In a linear semantic tree, one of the two branches of each node leads to a failure node. Such linearity is attractive because it enables us to focus our efforts on closing the remaining branch. Unfortunately, the strategy of building a closed linear semantic tree is incomplete. To help to achieve closure, we introduce resolutions to the clauses when necessary. This in turn leads to a strategy that combines the construction of semantic trees with resolution-refutation.

Next, we present a parallel semantic tree theorem prover, called PrHERBY, combined with the resolution-refutation method. Whereas many parallel systems adopt a competitive model, in which each slave processor runs with different parameters or strategies, that is therefore limited to a number of heuristics available [Sch97, New98, AN98], our parallel system generally shows extensive scalability by strategically selecting atoms with the help of dedicated resolutions. Moreover, a parallel grounding scheme allows each system to have its own instance of generated atoms, which increases the likelihood of success. The parallel system presented here shows better performance and generally constructs shorter semantic trees than HERBY and its parallel version, PHERBY.

1.1 Test sets

Producing a list of problems to test theorem provers is hard, because what is easy for one system might not be for another. Many people have tried to compile lists of theorems that can evaluate provers fairly. Among these is the Stickel test set, compiled to test his PTTP (Prolog Technology Theorem Prover) in 1988 [Sti88]. It receives wide recognition and provides a suitable testing environment because of its wide range of theorems with varying degrees of difficulty. Besides PTTP, it has been used for decades to develop and test theorem provers, such as SETHEO [Let92]. It was also used to develop several strategies of HERBY and tune the prover to handle a variety of situations.

The set is too easy, however, for recent, highly sophisticated provers. In our experiment for this thesis, we used a subset of the TPTP (Thousands of Problems for Theorem Provers) [SS98] library, which is a large source of theorems developed to make the testing and evaluation of automated theorem proving systems more meaningful.

1.2 Organization of the thesis

In this section, we present an overview of the thesis. We have developed a system named PrHERBY (Parallel HERBY with Resolution), which embodies the proposed ideas. We will compare the performance of PrHERBY with the existing theorem provers, HERBY and its parallel version, PHERBY. We made experiments with the 420 CADE-14 selection lists, which is a part of the TPTP library and listed in Appendix A. Theorems used in examples are drawn mostly from the Stickel test set [Sti88].

In Chapter 2, we introduce theorem proving, preliminaries and present Herbrand's theorem. We present two examples to show how automated theorem proving is used in the real context of solving interesting problems in the field of artificial intelligence and mathematics. Next, we introduce various theorem proving procedures with examples such as semantic tree, resolution-refutation and linear refinement.

In Chapter 3, we introduce the linear form of semantic trees. The linear semantic tree is the formalization of the observation that many closed semantic trees tend to be very thin rather than complete. We show an example of integrating semantic trees with resolutions to help to achieve closure.

In Chapter 4, we present a *semantic tree with ungrounded atoms* and the correspondence relation with the *resolution proof tree*. We illustrate the difficulties using ungrounded atoms in constructing semantic trees through examples and present the parallel chained grounding strategy. We also discuss modified atom selection heuristics, and introduce the parallel unit list passing strategy.

In Chapter 5, we describe several previously developed parallel systems and discuss our new technique to achieve parallelism using iterative deepening depth-first search as a means of generating atoms. We show how to generate parallel semantic trees with a simple example. We present our implementation named PrHERBY. Then, we explain algorithms and operations of master processor and slave processors.

In Chapter 6, we present the results of experiments using variable number of machines. Instead of the Stickel test set, we choose problems in the TPTP library that were used in the CADE-14 competition. We show performance improvement by the parallel chained-resolution grounding scheme. We compare the performance of PrHERBY with HERBY and PHERBY. Also, we compare the depth of semantic trees between HERBY versus PrHERBY and PHERBY versus PrHERBY.

In Chapter 7, we discuss the scalability of PrHERBY. We measure the system times of master processor and the ratio of the generated versus used atoms. We compare the number of clauses generated by the master and slave processors to analyze the behavior of PrHERBY. Finally, a section is devoted to the review of semantic tree generation and resolution-refutation methods we discussed.

In Chapter 8, we present a summary along with some possible extensions and ideas for future work.

Chapter 2

Theorem Proving Procedures

Theorem proving techniques can provide valuable assistance in solving a wide variety of problems. These include answering open questions from mathematics, designing and validating logic circuits, and proving that computer programs meet their specifications. Theorem proving techniques have been used even in an application associated with computer vision [GWY99].

Finding a general decision procedure to check the validity of formulas of the first-order logic is one of the fundamental questions in mathematical logic. As shown by Church, the problem is undecidable. Although the whole first-order logic is undecidable, there are proof procedures that can verify whether a formula is valid, if indeed it is valid.

2.1 Preliminaries

In this thesis, we present theorems in Skolemized clause form in the language of first-order predicate calculus. The following terminology and definitions are used throughout the thesis and are not covered separately [CL73, New01].

Logical operators are: $\wedge$ [conjunction], $\vee$ [disjunction], $\neg$ [negation], $\Rightarrow$ [implication], $\Leftrightarrow$ [if and only if].

Quantifiers are: $\forall$[universal quantifier], $\exists$[existential quantifier].

A *term* is an expression composed of individual constants, variables, and functions that are themselves terms. For example, a, $f(x)$, and $g(f(y), b, f(b))$ are terms, given that a and b are constants, f and g are functions and x and y are variables.

A *predicate* is a relation in the domain of discourse. The relation is either true or false. A predicate has zero or more arguments that are terms.

If P is an n-place predicate symbol, and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an *atom*. This definition differs from the convention that an atom is usually a ground instance. In this thesis, we make a distinction between an ungrounded and a ground atom if necessary.

A *literal* is an atom or the negation of an atom; the negation is denoted by a negation symbol, $\neg$, preceding the predicate. A literal and its negation are called *complementary literals*.

A *formula* is defined as follows:

1. A literal is a formula.
2. If P and Q are formulas, then so are:
   $\neg P$, $P \vee Q$, $P \wedge Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$.
3. If P is a formula, for any variable x, so are: $(\forall x)P$, $(\exists x)P$.

A *clause* is a disjunction of literals. A clause containing only one literal is called a *unit clause*. When a clause contains no literals, it is an *empty* or a *null* clause and is denoted by Φ. The Φ clause indicates the logical constant FALSE, since it has no literal that can be satisfied by an interpretation.

Skolemization is a procedure to replace all the existentially quantified variables by Skolem functions in order to obtain a quantifier-free first order formula. For example, in the following formula,

$$(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)\ P(x, y, z, u, v)$$

the Skolemization procedure replaces x by a Skolem constant SK, u by a Skolem function $SK1(y, z)$ and obtains the Skolemized formula.

$$(\forall y)(\forall z)(\forall v)\ P(SK, y, z, SK1(y, z), v)$$

Skolem functions express the dependency of existentially quantified variables on those universally quantified variables placed before them.

A *set S of clauses* is a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

A *substitution* is a finite set of the form $\{t_1/v_1, t_2/v_2, \ldots, t_n/v_n\}$, where every v_i is a variable, every t_i is a term different from v_i and no two elements in the same set have the same variable after the stroke symbol. For example, a clause $P(x, f(a, y))$ with a substitution $\{b/x, c/y\}$ generates an *instance* $P(b, f(a, c))$ of the clause.

Let $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ be a substitution and E be an expression. Then $E\theta$ is an expression obtained from E by replacing simultaneously each occurrence of the variable v_i, $1 \leq i \leq n$, in E by the term t_i. $E\theta$ is called an *instance* of E.

A substitution θ is called a *unifier* for a set $\{E_1, \ldots, E_k\}$ if and only if $E_1\theta = E_2\theta = \cdots = E_k\theta$. The set is said to be *unifiable* if there is a unifier for it.

A substitution σ is *more general* than substitution θ if there is some substitution λ such that $\theta = \sigma\lambda$.

A unifier σ for a set $\{E_1, \ldots, E_k\}$ of expressions is a *most general unifier* if and only if there is a substitution λ such that $\theta = \sigma\lambda$ for each unifier θ for the set.

If two or more literals (with the same sign) of a clause C have a most general unifier σ, then $C\sigma$ is called a *factor* of C.

A clause C_1 *subsumes* a clause C_2 if there exists a substitution α such that $C_1\alpha$ is a subset of C_2. In this case, C_2 is a logical consequence of C_1.

The *most general unifier* can be defined also in connection with subsumption.

The *most general unifier(mgu)* of two predicate instances P_1 and P_2 is the one that produces a substitution instance P_3 such that P_3 subsumes every other substitution instances. The *mgu* produces the most general substitution instance.

Given two clauses C_1 and C_2 with no variables in common and two literals L_1 and L_2 in C_1 and C_2, respectively, if L_1 and $\neg L_2$ have a most general unifier σ, then the clause

$$(C_1\sigma - L_1\sigma) \vee (C_2\sigma - L_2\sigma)$$

is called a *binary resolvent* of C_1 and C_2. The literals L_1 and L_2 are called the *literals resolved upon*.

A *resolvent* of clauses C_1 and C_2 is one of the following binary resolvents:

1. a binary resolvent of C_1 and C_2.
2. a binary resolvent of C_2 and a factor of C_1.
3. a binary resolvent of C_1 and a factor of C_2.
4. a binary resolvent of a factor of C_1 and a factor of C_2.

Suppose two or more identical instances of a literal appear in the binary resolvent of two clauses. All but one of these identical instances can be deleted without changing the meaning of the binary resolvent. The resulting binary resolvent is called a *merge clause*.

A *set-of-support* of a set S of clauses is a subset T of S if $S - T$ is satisfiable. A set-of-support resolution is a resolution of two clauses that are not both from $S - T$.

An *interpretation* is an assignment of truth values to atoms in propositional logic. In first-order logic, an interpretation of a set of clauses consists of a nonempty domain D and an assignment of values to each constant, function and predicate symbol. For a constant a, an interpretation I assigns an element of D to a. For a function f, I assigns a mapping of D^n to D where n is the arity of f. (Note that $D^n = \{(x_1, \cdots, x_n) | x_1 \in D, \cdots, x_n \in D\}$). For an n-place predicate P, I assigns a mapping of D^n to $\{$TRUE,FALSE$\}$.

Finally, we explain two equality-related definitions.

Demodulation is the rewriting of terms. Given a clause C containing a term t and a unit equality clause of the form $Equal(\alpha, \beta)$, where t is an instance of α ($t = \alpha\sigma$), replace C by C', obtained from C by replacing all occurrences of t by t' where

$t' = \beta\sigma$. For example, given $P(f(a,a))$ and demodulator $Equal(f(x,x),g(x))$, the demodulation replaces $P(f(a,a))$ with $P(g(a))$.

Paramodulation is a generalization of demodulation. Given a clause C containing a term t and a clause D containing an equality literal Equal(α,β), where t unifies with α with substitution μ, we derive clause C', which is $C\mu$ with t replaced by $\beta\mu$. For example,

$D: Equal(sum(0,x),x)$	$\alpha = sum(0,x); \beta = x$
$C: Equal(sum(y,minus(y)),0)$	$t = sum(y,minus(y))$
C': $Equal(minus(0),0)$	$\mu: \{0/y, minus(0)/x\}$

Before proceeding, we define a logical consequence, validity and unsatisfiability of a formula formally as follows.

Definition 1 Given formulas $F_1, \ldots, F_n$ and a formula G, G is said to be a *logical consequence* of $F_1, \ldots, F_n$ (or G *logically follows* from $F_1, \ldots, F_n$) if and only if for any interpretation I in which $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ is true, G is also true.

Definition 2 A formula is said to be *valid* if and only if it is true under all interpretations. A formula is said to be *invalid* if and only if it is not valid.

Definition 3 A formula is said to be *inconsistent* or *unsatisfiable* if and only if it is false under all interpretations. A formula is said to be *consistent* or *satisfiable* if and only if it is not inconsistent.

If G is a logical consequence of axioms $F_1 \wedge F_2 \wedge \ldots \wedge F_n$, the formula $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ is called a *theorem*. In mathematics as well as in other fields, we can formulate many problems as formulas that consist of axioms, hypotheses and conjectures. Moreover, it can be shown that proving that a formula is a logical consequence of a finite set of formulas is equivalent to proving that the formula is *valid* or that the negation of the formula is *inconsistent* [CL73]. Specifically, in a *refutation* theorem prover, one can prove the *unsatisfiability* (*inconsistency*) of a theorem by negating the given formula i.e., $\neg(F_1 \wedge F_2 \wedge \ldots \wedge F_n \Rightarrow G)$, and showing that this yields a contradiction. Hereafter, *resolution* refers to *resolution-refutation* unless otherwise stated.

The proofs generated by theorem provers often show precisely how and why conjectures follow from sets of axioms and hypotheses. The proof sequence might not only be a convincing argument that the conjecture is a logical consequence of axioms and hypotheses, but may show the process leading to problem solutions. For example, in the missionaries and cannibals problem below, the proof would describe safe moves for missionaries to cross the river.

Example 1 (The missionaries and cannibals problem) Three missionaries and three cannibals stand on the left bank of a river. They all want to get to the right side. They have a boat that can hold one or two of them at a time. If at any time the cannibals outnumber the missionaries on either side of the river, however, they will eat the missionaries. Is it possible for all six to cross the river without losing a missionary?

Solution : We define a predicate S(i,j,k) representing a state where i is the number of missionaries on the left side of the river, j is the number of cannibals on the left side of the river, and k denotes the location of the boat (R: right side, L: left side).

The initial state is S(3,3,L) and the goal state S(0,0,R). All the possible moves can be defined as in the next page. The first axiom,

$$S(3,3,L) \Rightarrow \{S(3,1,R) \wedge S(3,2,R) \wedge S(2,2,R)\}$$

indicates how 3 missionaries and 3 cannibals on the left side of the bank with the boat possibly move. In fact, the axiom is a conjunction of the 3 axioms presented below.

$$S(3,3,L) \Rightarrow \{S(3,1,R)\}$$

$$S(3,3,L) \Rightarrow \{S(3,2,R)\}$$

$$S(3,3,L) \Rightarrow \{S(2,2,R)\}$$

If the formulated problem is submitted to THEO [New01] after negating the goal statement, it generates the following proof. For convenience, parentheses are ignored.

Hypothesis $S(3,3,L)$

Axioms

$$S(3,3,L) \Rightarrow \{S(3,1,R) \wedge S(3,2,R) \wedge S(2,2,R)\}$$
$$S(3,1,R) \Rightarrow \{S(3,2,L) \wedge S(3,3,L)\}$$
$$S(3,2,R) \Rightarrow S(3,3,L)$$
$$S(2,2,R) \Rightarrow \{S(3,2,L) \wedge S(3,3,L)\}$$
$$S(3,2,L) \Rightarrow \{S(3,1,R) \wedge S(3,0,R) \wedge S(2,2,R)\}$$
$$S(3,0,R) \Rightarrow \{S(3,1,L) \wedge S(3,2,L)\}$$
$$S(3,1,L) \Rightarrow \{S(3,0,R) \wedge S(1,1,R)\}$$
$$S(1,1,R) \Rightarrow \{S(2,2,L) \wedge S(3,1,L)\}$$
$$S(2,2,L) \Rightarrow \{S(1,1,R) \wedge S(0,2,R)\}$$
$$S(0,2,R) \Rightarrow \{S(2,2,L) \wedge S(0,3,L)\}$$
$$S(0,3,L) \Rightarrow \{S(0,2,R) \wedge S(0,1,R)\}$$
$$S(0,1,R) \Rightarrow \{S(0,2,L) \wedge S(1,1,L)\}$$
$$S(0,2,L) \Rightarrow \{S(0,1,R) \wedge S(0,0,R)\}$$
$$S(1,1,L) \Rightarrow \{S(0,0,R) \wedge S(0,1,R)\}$$

Conjecture $S(0,0,R)$

THEO finds a proof in phase 1 and reconstructs the proof in phase 2. A proof line, for example, `32>(29a,26b) S00R ~S01R` shows that clause number 32 was generated by resolving the first literal (written by `a`) of clause 29 with the second literal (written by `b`) of clause 26.

Phases 1 and 2 clauses used in proof:

```
32>(29a,26b) S00R ~S01R
 33>(32b,24b) S00R ~S03L
  34>(33b,22b) S00R ~S02R
   35>(34b,20b) S00R ~S22L
    36>(35b,17b) S00R ~S11R
     37>(36b,16b) S00R ~S31L
      38>(37b,13b) S00R ~S30R
       39>(38b,11b) S00R ~S32L
        40>(39b,5b)  S00R ~S31R
         41>(40b,2b)  S00R ~S33L
          42>(41b,1a)  S00R
           43>(42a,31a) []
```

The theorem prover is attempting to prove that the negated conjecture statement together with the hypothesis and axioms is unsatisfiable. The conjecture statement $S(0,0,R)$ is obtained by taking off $\neg S(3,3,L)$ from the proof line

```
41>(40b,2b)   S00R ~S33L
```

by resolving with the first clause $S(3,3,L)$. Consequently, the reverse order, showing how $\neg S(3,3,L)$ is reached, gives a safe sequence for the missionaries and the cannibals to cross the river.

The next example shows an application of theorem proving in mathematics. For some of the open questions in mathematics, theorem provers helped to solve them by supplying a proof, by generating a finite model, or by finding a counterexample [WOLB92]. The following example shows also the importance of demodulation and paramodulation strategies for solving mathematical theorems.

Example 2 In a group, $(x^{-1})^{-1} = x$ for all x in the group [LO85].

Axioms for a group:

1. $Equal(f(e,x),x)$	e is a left identity
2. $Equal(f(x,e),x)$	e is a right identity
3. $Equal(f(g(x),x),e)$	there exists a left inverse
4. $Equal(f(x,g(x)),e)$	there exists a right inverse
5. $Equal(f(f(x,y),z),f(x,f(y,z)))$	associativity
6. $Equal(x,x)$	reflexivity of equality
7. $\neg Equal(g(g(a)),a)$	denial of the theorem

Proof:

8. $Equal(z,f(x,f(g(x),z)))$

The clause is deduced by paramodulating *into* term $f(x,y)$ of clause 5 using *from* term $f(x,g(x))$ of clause 4 resulting in

$$Equal(f(e,z),f(x,f(g(x),z)))$$

Demodulating into the term $f(e,z)$ of the result using clause 1 will generate clause 8.

9. $Equal(g(g(x)),x)$

This clause was deduced by paramodulating *from* term $f(x,g(x))$ of clause 4 *into* term $f(g(x),z)$ of clause 8 resulting in

$$Equal(g(g(x)),f(x,e))$$

$f(x,e)$ is x by demodulation, using clause 2.

10. Φ

Clauses 9 and 7 resolve together, thus proving the theorem.

2.2 Semantic trees

By definition, a set S of clauses is unsatisfiable if and only if it is false under all interpretations over all domains. Since considering all interpretations over all domains is hard for most theorems of any difficulty, we can attempt to find a single special domain $\mathcal{H}$ such that $\mathcal{S}$ is unsatisfiable if and only if it is false under all interpretations over this particular domain. Such a domain does exist, and it is known as the *Herbrand universe* of $\mathcal{S}$. This is defined as follows:

Definition 4 *(Herbrand universe)* Let $\mathcal{H}_0$ be the set of constants appearing in $\mathcal{S}$. If no constant appears in $\mathcal{S}$, then $\mathcal{H}_0 = \{a\}$, where a is an arbitrary constant. For $i = 0, 1, 2, ...$, let $\mathcal{H}_{i+1}$ be the union of $\mathcal{H}_i$ and the set of all terms of the form $f^n(t_1, ..., t_n)$ for all n-place functions f^n occurring in $\mathcal{S}$, where t_j, $j = 1, ..., n$, are members of the set $\mathcal{H}_i$. Then each $\mathcal{H}_i$ is called the i-level constant set of $\mathcal{S}$, and $\mathcal{H}_\infty$ is called the Herbrand universe of $\mathcal{S}$ [CL73].

Herbrand's Theorem states that there is a finite set of instantiations of clauses in $\mathcal{S}$ that is propositionally unsatisfiable if a set $\mathcal{S}$ of clauses is unsatisfiable. Herbrand's Theorem enables us to handle a potentially infinite domain with finite means.

When no variable appears in a term, a set of terms, an atom, a set of atoms, a literal, a clause, or a set of clauses, they are called *ground* ones. Thus, we can use a ground term, a ground atom, a ground literal, and a ground clause to indicate that no variable occurs in them.

Definition 5 *(Herbrand base)* Let $\mathcal{S}$ be a set of clauses. The set of ground atoms of the form $P^n(t_1, ..., t_n)$ for all n-place predicates P^n occurring in $\mathcal{S}$, where $t_1, ..., t_n$ are elements of the Herbrand universe of $\mathcal{S}$, is called the Herbrand base of $\mathcal{S}$, or the *atom set*. Elements of the Herbrand base are called ground atoms.

Example 3 Consider theorem WOS12 (from the Stickel test set). There are three predicates: p, $Equal$, and o; two functions: f and g; and two constants: a and e.

1. $p(e, x, x)$
2. $p(g(x), x, e)$
3. $p(x, y, f(x, y))$

4. $\neg p(x,y,u) \vee\ p(y,z,v) \vee\ p(u,z,w) \vee p(x,v,w)$

5. $\neg p(x,y,u) \vee\ p(y,z,v) \vee\ p(x,v,w) \vee p(u,z,w)$

6. $Equal(x,x)$

7. $\neg Equal(x,y) \vee Equal(y,x)$

8. $\neg Equal(x,y) \vee \neg Equal(y,z) \vee Equal(x,z)$

9. $\neg p(x,y,u) | \neg p(x,y,v) \vee Equal(u,v)$

10. $\neg Equal(u,v) \vee \neg p(x,y,u) \vee p(x,y,v)$

11. $\neg Equal(u,v) \vee \neg p(x,u,y) \vee p(x,v,y)$

12. $\neg Equal(u,v) \vee \neg p(u,x,y) \vee p(v,x,y)$

13. $\neg Equal(u,v) \vee Equal(f(x,u), f(x,v))$

14. $\neg Equal(u,v) \vee Equal(f(u,y), f(v,y))$

15. $\neg Equal(u,v) \vee Equal(g(u), g(v))$

16. $p(x,e,x)$

17. $p(x,g(x),e)$

18. $\neg o(x) \vee \neg o(y) \vee \neg p(x,g(y),z) \vee o(z)$

19. $\neg o(x) \vee \neg Equal(x,y) \vee o(y)$

20. $o(a)$

Negated_conclusion

21. $\neg o(e)$

The Herbrand universe of the theorem is the infinite set:

Herbrand universe(WOS12) = $\{a, e, g(a), g(e), f(a,a), f(a,e), f(e,a),$ $f(e,e), g(g(a)), g(g(e)), g(f(a,a)), g(f(a,e)), g(f(e,a)), g(f(e,e)), f(a,g(a)),$ $f(a,g(e)), f(a,g(f(a,a)), \cdots\}$.

The Herbrand base of the theorem is then:

Herbrand base(WOS12) = $\{o(a), Equal(a,a), p(a,a,a), o(e), Equal(a,e),$

$Equal(e,a), Equal(e,e), p(a,a,e), p(a,e,a), p(e,a,a), p(e,a,e), p(e,e,a),$
$p(e,e,e), o(g(a)), Equal(a,g(a)), Equal(e,g(a)), \cdots\}$.

Definition 6 *(Ground instance)* A ground instance of a clause C of a set $\mathcal{S}$ of clauses is a clause obtained by replacing variables in C by members of the Herbrand universe of $\mathcal{S}$.

We now consider a special interpretation over the Herbrand universe.

Definition 7 *(Herbrand interpretation)* Let $\mathcal{S}$ be a set of clauses; $\mathcal{H}$, the Herbrand universe of $\mathcal{S}$; and $\mathcal{I}$, an interpretation of $\mathcal{S}$ over $\mathcal{H}$. $\mathcal{I}$ is said to be an Herbrand interpretation of $\mathcal{S}$ if it satisfies the following conditions:

1. To each constant, map all constants in $\mathcal{S}$ to themselves.
2. Let f be an n-place function and $(h_1, \ldots, h_n)$ be elements of $\mathcal{H}$. In $\mathcal{I}$, f is assigned a function that maps $(h_1, \ldots, h_n)$ (an element of $\mathcal{H}^n$) to $f(h_1, \ldots, h_n)$ (an element of $\mathcal{H}$).

A Herbrand interpretation is conveniently represented by a set

$$\mathcal{I} = \{m_1, m_2, \ldots\}$$

where m_j is either A_j or $\neg A_j$ of an atom set $A = \{A_1, A_2, \ldots\}$ for $j = 1, 2, \ldots$.

Theorem 1 *(Herbrand's Theorem, Version II)* A set $\mathcal{S}$ of clauses is unsatisfiable if and only if there is a finite unsatisfiable set $\mathcal{S}'$ of ground instances of clauses of $\mathcal{S}$. (see the proof in [CL73].)

Even though it is true that we need to consider only Herbrand interpretations to check unsatisfiability, the Herbrand interpretations are infinite if a set of clauses contains functions because the possible Herbrand universe is infinite. Herbrand's theorem has a particular structure, semantic tree, which allows us to express unsatisfiability of a set $\mathcal{S}$ in a systematic way. The semantic trees we shall consider here are *binary semantic trees*.

Definition 8 *(Semantic Tree)* Given a set $\mathcal{S}$ of clauses, a semantic tree is a downward-growing binary tree in which the branches are labelled with atoms from the Herbrand base (HB) and their negations ($\neg HB$) (see Figure 2.1). Each node N_j ($j = 1, 2, 3, \ldots$) is assigned a set of clauses as follows:

- The root node N_1 is assigned $\mathcal{S}$.
- For any other node N_j ($j = 2, 3, \ldots$) with the nodes on the path to it, namely $N_1, \ldots, N_{j-1}$ and with the branches leading to it labelled with HB or $\neg HB$, we assign all resolvents of HB or $\neg HB$ with all clauses in the nodes $N_1, \ldots, N_{j-1}$ and with the clauses so generated except resolvents already generated in the nodes on the path to it or in N_j.
- The edges below any node are labelled with complementary literals.

Example 4 Consider theorem S16QW (from the Stickel test set).

1 : $p(x, a) \vee p(x, f(x))$

2 : $p(x, a) \vee p(f(x), x)$

Negated_conclusion

3 : $\neg p(x, a) \vee \neg p(x, y) \vee \neg p(y, x)$

Figure 2.1 shows a corresponding semantic tree of the theorem S16QW. The root node contains the given set $\mathcal{S}$ of clauses. Each other node contains resolvents of an atom on the branch leading to it with all clauses in the nodes on the path.

For each node N, let $I(N)$ be the union of all the atoms attached to the edges connecting the root of the tree with the node N and $\mathcal{A}$ be the atom set of $\mathcal{S}$. A node N is called a *failure node* if $I(N)$ falsifies some ground instance of a clause in $\mathcal{S}$, and there is no other failure node on the path from N to the root of the tree. A semantic tree is *closed* if and only if every branch terminates in a *failure node.* In that case, the set $\mathcal{S}$ is unsatisfiable. A *complete semantic tree* is a semantic tree in which every path from the root node down the tree contains every atom or negated atom of the set $\mathcal{A}$.

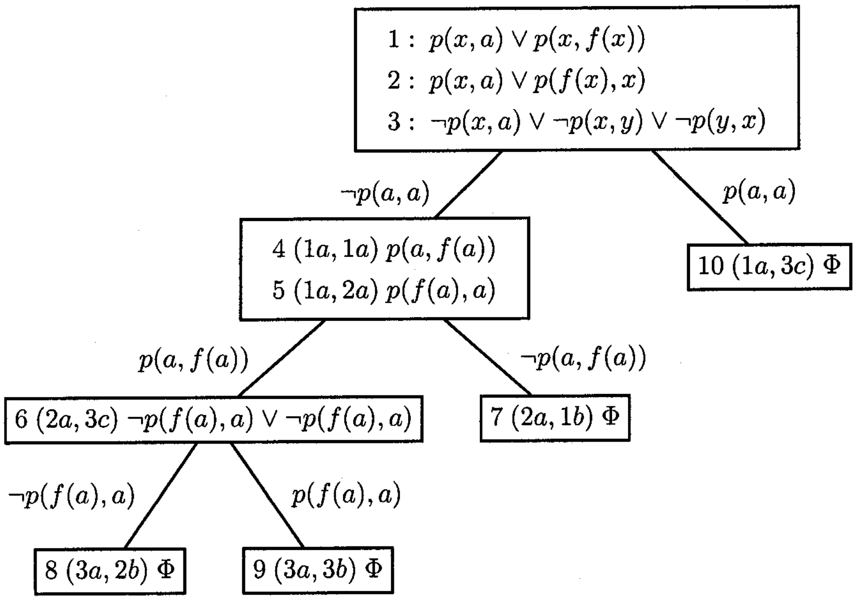


Figure 2.1: A binary semantic tree of theorem S16QW

The mechanism of constructing a semantic tree allows insight into the process of establishing unsatisfiability for a set of clauses. The Herbrand base of an unsatisfiable set S of clauses can be infinite with the corresponding infinite complete semantic tree. However, we can build a closed semantic tree with only a finite subset of the Herbrand base if indeed it is unsatisfiable.

2.3 Resolution-refutation proofs

The resolution principle was first proposed by J.A. Robinson in 1965 [Rob65] and constituted a major breakthrough for automated theorem proving. Resolution is a binary operation inferring a clause from two clauses. Typical resolution-refutation provers generate such a new clause from a given set S of clauses and attempt to produce the empty clause Φ. If S contains Φ, then S is unsatisfiable. Otherwise, S must be examined to see whether Φ can be derived.

In the previous section, we presented semantic trees. Now, we shall derive resolution-refutation proofs from the semantic tree to prove the completeness of the resolution principle.

Given a semantic tree $\mathcal{T}$, some non-failure nodes of the $\mathcal{T}$ can be forced to become failure nodes if new resolvents of clauses in $\mathcal{S}$ are added to $\mathcal{S}$. Thus, the number of nodes in $\mathcal{T}$ can be reduced and Φ will eventually be derived. Based on the semantic tree, the algorithm for obtaining a resolution-refutation proof repeatedly

1. finds two failure nodes that are siblings
2. resolves the clauses that fail at these nodes to generate additional clauses of the resolution-refutation proof
3. generates a new semantic tree based on the set of base clauses and those clauses that were added in step 2.

When Φ is generated, the algorithm terminates outputting a resolution-refutation proof. In the example below, we show the procedure that produces a resolution-refutation proof from a closed semantic tree.

Example 5 Consider the closed semantic tree for theorem S16QW (Figure 2.1). In the semantic tree, we choose the two failure nodes containing clauses 8 and 9 and resolve together clauses 2 and 3, which fail at these nodes. These two falsified clauses must have a complementary pair of literals, and therefore can be resolved. Before resolving the two clauses, clause 3 is factored. This generates clause 4. Hence, the resolution occurs between clauses 2 and 4:

$4:\ (3ab)\ \neg p(x,a) \vee \neg p(a,x)$

$5:\ (2b,4a)\ p(a,a) \vee \neg p(a,f(a))$

Next, construct the semantic tree for the modified set of clauses after adding clauses 4 and 5 (Figure 2.2). Again, we choose the two failure nodes containing clauses 9 and 10 and resolve together clauses 5 and 1, which fail at these nodes. In this case, resolvent 6 is factored and clause 7 added:

$6:\ (5b,1b)\ p(a,a) \vee p(a,a)$

$7:\ (6ab)\ p(a,a)$

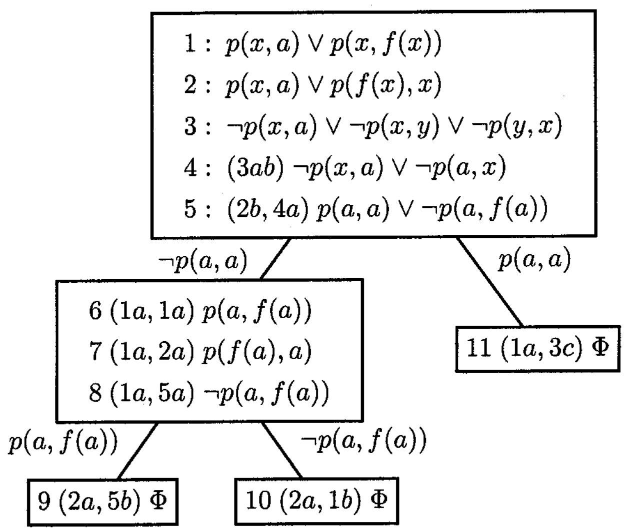


Figure 2.2: Modified semantic tree for S16QW after adding clauses 4 and 5

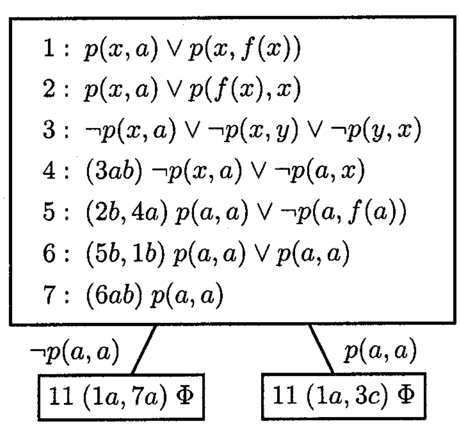


Figure 2.3: Modified semantic tree for S16QW after adding clauses 6 and 7

$$\begin{array}{l}
1: \; p(x,a) \vee p(x,f(x)) \\
2: \; p(x,a) \vee p(f(x),x) \\
3: \; \neg p(x,a) \vee \neg p(x,y) \vee \neg p(y,x) \\
4: \; (3ab) \; \neg p(x,a) \vee \neg p(a,x) \\
5: \; (2b,4a) \; p(a,a) \vee \neg p(a,f(a)) \\
6: \; (5b,1b) \; p(a,a) \vee p(a,a) \\
7: \; (6ab) \; p(a,a) \\
8: \; (7a,4a) \; \neg p(a,a) \\
9: \; (7a,8a) \; \Phi
\end{array}$$

Figure 2.4: The root node of the semantic tree for S16QW after adding clauses 8 and 9

After constructing the semantic tree for the modified set of clauses with added clauses 6 and 7 (Figure 2.3), we choose the remaining two failure nodes and resolve together clauses 7 and 3 that fail at these nodes. Clause 3 is factored again and the previously generated clause 4 is used for the resolution:

$8: \; (7a,4a) \; \neg p(a,a)$

The semantic tree constructed for the modified set of clauses after adding clause 8 finally generates:

$9: \; (7a,8a) \; \Phi$

This "collapsing" of the semantic tree actually corresponds to a resolution proof which is presented with clauses 4 to 9 in Figure 2.4.

2.4 Linear resolution

Linear resolution was proposed by D.W. Loveland [Lov70] and D. Luckham [Luc70] in 1970. Among the most powerful refinements of unrestricted resolution, linear resolution offers exceptional opportunities for the application of heuristic searches by virtue of relatively simple structure of the linear proof. Furthermore, every theorem has a linear resolution proof.

Definition 9 Given a set $\mathcal{S}$ of clauses, a proof beginning with a clause C_0 in $\mathcal{S}$ is a *linear resolution* if the following conditions are met as shown in Figure 2.5:

- for $i = 0, 1, \ldots, n-1$, C_{i+1} is a resolvent of C_i (called a center clause) and B_i (called a side clause), and
- each B_i is either in $\mathcal{S}$, or is a C_j for some $j, j < i$.

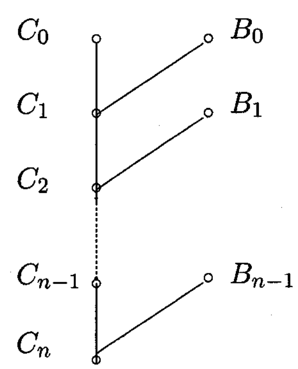


Figure 2.5: Linear resolution

Example 6 Consider theorem S16QW in Figure 2.6. A linear resolution proof is shown in Figure 2.7. A clause can be used more than once in the proof.

$$1:\ p(x,a) \vee p(x,f(x))$$
$$2:\ p(x,a) \vee p(f(x),x)$$
$$\mathit{negated\_conclusion}$$
$$3:\ \neg p(x,a) \vee \neg p(x,y) \vee \neg p(y,x)$$

Figure 2.6: Theorem S16QW

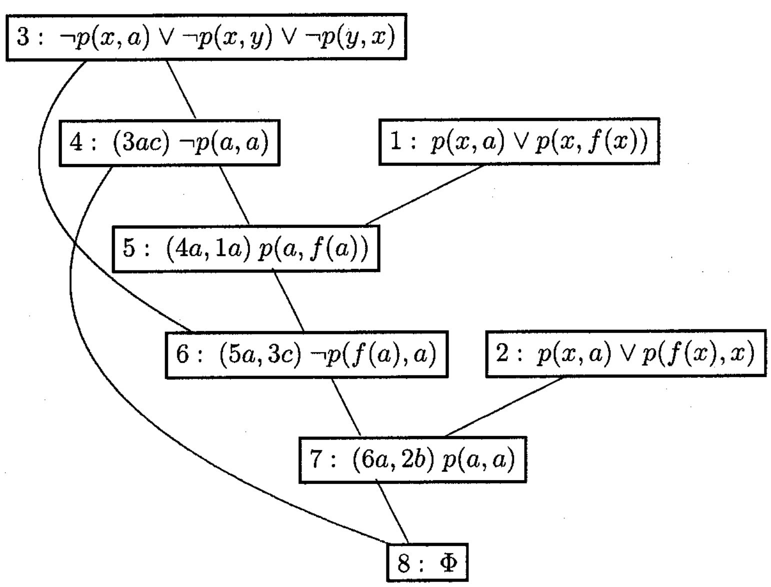


Figure 2.7: A linear resolution proof of theorem S16QW

Chapter 3

Semantic Trees with Resolutions

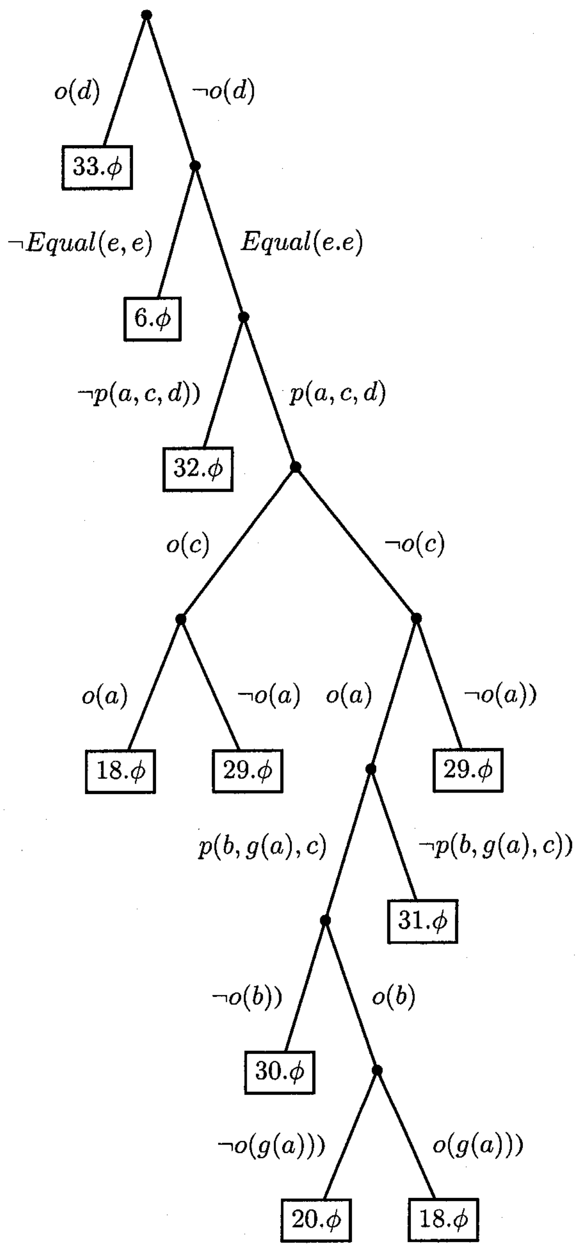
In this chapter, we consider linear properties of semantic trees and then define linear semantic trees integrated with resolutions. With this observation, we introduce a strategy that constructs semantic trees with resolutions to facilitate closure of the trees.

3.1 Motivation

We introduced resolution-refutation proofs in section 2.3. There is a procedure to convert a semantic tree to a resolution proof by successively collapsing a pair of leaves. It is possible also to convert a resolution proof to a semantic tree (discussed in section 4.1). We also introduced linear resolution in section 2.4. This, then, raises the question: If there is a linear resolution proof, "*is there not also a closed linear semantic tree?* "

We made an observation that many semantic trees found by HERBY are rather thin. An experiment clarified this by showing that more than half of the proofs are almost linear, which means all non-terminal nodes are on one path [YAN98]. The experiment showed that 33 out of the 78 proofs found by HERBY in the Stickel test set are completely linear; another 8 had all but one non-terminal node on a single path; and yet another 8 had all but two non-terminal nodes on a single path. The following example shows a typical semantic tree. It is linear with the exception of one branch.

Example 7 Consider theorem WOS19 from the Stickel test set. The closed semantic tree is linear except for a single branch.



1. $p(e, x, x)$
2. $p(g(x), x, e)$
3. $p(x, y, f(x, y))$
4. $\neg p(x, y, u) \lor \neg p(y, z, v) \lor \neg p(u, z, w) \lor p(x, v, w)$
5. $\neg p(x, y, u) \lor \neg p(y, z, v) \lor \neg p(x, v, w) \lor p(u, z, w)$
6. $Equal(x, x)$
7. $\neg Equal(x, y) \lor Equal(y, x)$
8. $\neg Equal(x, y) \lor \neg Equal(y, z) \lor Equal(x, z)$
9. $\neg p(x, y, u) \lor \neg p(x, y, v) \lor Equal(u, v)$
10. $\neg Equal(u, v) \lor \neg p(x, y, u) \lor p(x, y, v)$
11. $\neg Equal(u, v) \lor \neg p(x, u, y) \lor p(x, v, y)$
12. $\neg Equal(u, v) \lor \neg p(u, x, y) \lor p(v, x, y)$
13. $\neg Equal(u, v) \lor Equal(f(x, u), f(x, v))$
14. $\neg Equal(u, v) \lor Equal(f(u, y), f(v, y))$
15. $\neg Equal(u, v) \lor Equal(g(u), g(v))$
16. $p(x, e, x)$
17. $p(x, g(x), e)$
18. $\neg o(x) \lor \neg o(y) \lor \neg p(x, y, z) \lor o(z)$
19. $\neg o(x) \lor \neg Equal(x, y) \lor o(y)$
20. $\neg o(x) \lor o(g(x))$
21. $o(e)$
22. $\neg Equal(u, v) \lor Equal(i(x, u), i(x, v))$
23. $\neg Equal(u, v) \lor Equal(i(u, x), i(v, x))$
24. $o(x) \lor o(y) \lor o(i(x, y))$
25. $o(x) \lor o(y) \lor p(x, i(x, y), y)$
26. $\neg p(x, u, z) \lor \neg p(x, v, z) \lor Equal(u, v)$
27. $\neg p(u, y, z) \lor \neg p(v, y, z) \lor Equal(u, v)$
28. $Equal(g(g(x)), x)$
29. $o(a)$
30. $o(b)$
31. $p(b, g(a), c)$
32. $p(a, c, d)$

negated_conclusion

33. $\neg o(d)$

3.2 Linear semantic trees

The linear property of semantic trees is as attractive as the linear property of resolution, because it allows us to focus our efforts on closing the remaining branch. Such linearity is the result of an attempt to close the tree as soon as possible. Hence, unit clauses are preferred over other non-unit clauses when it comes to atom selection heuristics. As a result, trees generally become thinner as proofs go deeper. In fact, unit preference or fewest-literals preference are the favored strategies in many theorem proving systems.

Let us define the linear semantic tree as follows:

Definition 10 *(Linear Semantic Tree)* Let N be a node at which an atom A is to be selected and $T(N)$ be the set of clauses assigned to nodes on the path from the root to N. A *linear semantic tree* for $T(N)$ is a semantic tree, where each edge is labelled with an atom or negated atom in such a way that:

- for the node N, there are two immediate edges E_1, E_2, with a positive atom A and its negative $\neg A$ attached to them. One of the atoms always falsifies a clause in the set $T(N)$ of clauses.
- for the node N, let $I(N)$ be the union of all the sets of atoms attached to the edges of non-terminal nodes from root to the N. $I(N)$ does not contain any complementary pairs.

The linear semantic trees defined above are semantic trees with all non-terminal nodes on one path. A linear semantic tree is *closed* if and only if every branch terminates in a *failure node*. Unfortunately, a linear semantic tree exists for some but not all theorems. This tree reminds us of a *vine-form* proof in linear resolution or *input resolution*. This adds a further restriction in that each resolution in the proof has at least one base clause as an input, but not every theorem has a proof in vine-form.

A semantic tree represents the possible Herbrand interpretations of the theorem. If the tree is linear as defined above, one of the two branches from each node in the semantic tree always leads to a failure node. This is equivalent to *unit resolution*, in which one of the two parent clauses is a unit clause. Unit resolution is known to be the same as *input resolution* [Cha70], and not every theorem has a proof of the input resolution.

1.$\neg j \vee a \vee h$
2. $k \vee h \vee j$
3.$\neg k \vee h \vee j$
4.$\neg a \vee \neg b$
5.$\neg a \vee b$
6.$\neg h \vee \neg c$
$negated\_conclusion$
7.$\neg h \vee c$

Figure 3.1: Theorem S06ANCES

Example 8 Consider theorem $S06ANCES$ from the Stickel test set in Figure 3.1. A linear semantic tree cannot be constructed for this theorem, because there is no way to choose an atom to close the first branch of the tree. A closed, non-linear semantic tree, however, is possible as shown in Figure 3.2.

The ■ in the figure denotes a *useless atom* [New01], which means that no resolvents are generated by the atom. There are two useless atoms in the tree. If an atom turns out to be useless, the right branch is not searched. This is one of the heuristics used in HERBY. It is justified, because the right subtree would be closed with the same atoms as the left.

To help to achieve linearity, we introduce resolution when necessary. If a given set S of clauses is a theorem, one must obtain successively shorter clauses to deduce a contradiction. Providing unit clauses through resolution gives a way of progressing rapidly toward shorter clauses. We call this a *combination strategy*.

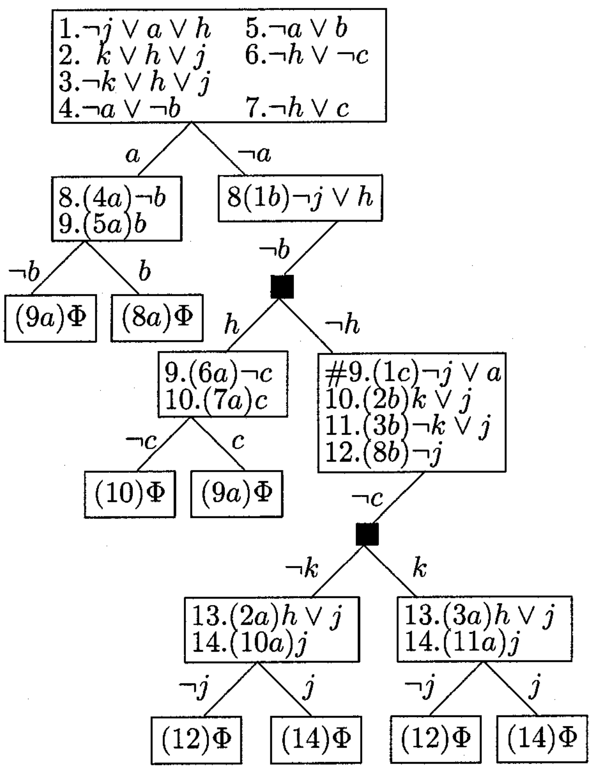


Figure 3.2: A closed semantic tree of theorem S06ANCES

Resolution and semantic tree methods are sound and complete. Neither the completeness nor the soundness of the combination strategy is violated whether the unit clauses come from a semantic tree or from resolution.

We can construct a closed linear semantic tree for the above theorem S06ANCES as shown in Figure 3.3. Two resolutions are added during the construction of the linear semantic tree. Notice that the atoms used in the proof are the same as the ones used in the closed semantic tree in Figure 3.2, except for the useless atoms b and c.

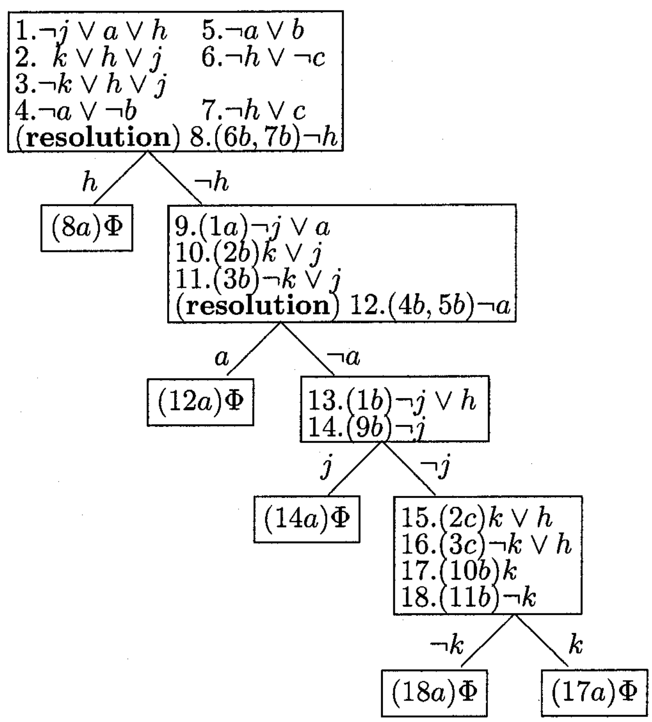


Figure 3.3: A closed linear semantic tree with resolutions of theorem S06ANCES

3.3 Semantic trees with resolutions

The idea of combining semantic trees with resolutions has a rudimentary origin back to HERBY: the BCR (base clause resolution) heuristic. HERBY resolves unit clauses in the set-of-support with every other clause, adding at most one resolvent for each input pair to the set of base clauses [New01]. That HERBY can solve 79 theorems out of the 84 Stickel test set is partially attributable to this strategy. Without the BCR heuristic, only 75 of the 84 are solved, since S05HASP2, S20FLEI1, S21FLEI2 and S29WOS5 cannot be solved within the 60-second time limit.

The combination strategy has been tried on several occasions [Lov69, Hur99]. In the case of GANDALF_TAC [Hur99], Gandalf was combined into a higher-order logic theorem prover to support first-order logic proofs. Most other combining strategies are used to improve performance, because there is a tendency for the two procedures to complement each other.

It is considered that combining several different tactics is apparently effective in the theorem proving community because different strategies work for different sets of theorems. Almulla [AN96] mentioned that there were some theorems for which the semantic tree prover performed better than the resolution-refutation prover and vice versa. Though the argument is no longer true in his examples as discussed in 7.4, combining semantic tree method with resolutions gives interesting prospect.

The usefulness of adding resolvents is due to the fact that a resolvent is a logical consequence of the resolved clauses. The resolvent is true for every interpretation in which both parent clauses are true.

The closed linear semantic tree in Figure 3.3 is simpler than the closed semantic tree in Figure 3.2, because the linear structure eliminates useless atoms. In the presence of useless atoms, we are not able to continue the construction of the linear semantic tree as they do not contribute to closure. On the other hand, if a unit clause is added and an atom is chosen from the unit clause, the linearity is guaranteed as at least one branch will be closed. This increases the importance of smart atom selection strategies, however, and that of finding proper resolution pairs.

With these observations, we envisioned a strategy in which resolutions are applied whenever necessary to construct semantic trees. If an atom is selected through a series of resolutions, the atom is more likely to trigger closure than an atom chosen arbitrarily from sets of clauses as shown in Figure 3.3. Moreover, we can apply several strategies developed for resolution-refutation to generate these atoms.

In our implementation of a parallel system based on master-slave, two different ideas with the previous discussions are applied. Firstly, resolutions are preformed solely in the master processor to generate atoms located deep inside of the theorem. Secondly, linearity is not enforced because the resolvents of the master are not added to the set of the clauses in a slave processor and a slave processor runs constructing a non-linear semantic tree. We are concerned mainly with the possibility of integrating resolutions into semantic trees and therefore, with the parallel strategy that was not limited by a number of available strategies.

We discuss grounding strategies and a parallel semantic tree prover that combines with resolutions in the next chapters.

Chapter 4

Grounding Strategies and atom selection heuristics

In this chapter, we introduce a semantic tree with ungrounded atoms and show a correspondence of the semantic tree with resolution-refutation. Then, we propose a strategy for grounding atoms useful in constructing semantic trees in parallel. We also discuss the modified atom selection heuristics.

The atoms of a semantic tree are ground ones that are a subset of the Herbrand base (Definition 5). However, most atom selection heuristics do not guarantee the absence of variables. Therefore, a grounding scheme must be used and good grounding schemes are often the key to proving difficult theorems. However, it may result in delayed or missed closure if not properly applied. For that reason, many atom selection heuristics are designed to return atoms with a minimum number of variables, thus minimizing the effect of the grounding procedure.

HERBY approaches the problem by simply assigning the constants given in a theorem to variables. It uses an arbitrary constant a if no constant appears in the theorem. If a theorem has $n > 1$ constants, HERBY grounds a literal formed at ply i by selecting the $(i+1)^{th}$ **modulo** n constant. We call this a *fixed grounding strategy*. Resolution, on the contrary, uses the most general unifier to generate resolvents. To explain this, the following theorem is relevant here.

Theorem 2 If C is a resolvent of $C1$ and $C2$ and if $D1$ and $D2$ subsume $C1$ and $C2$, respectively, then there is a resolvent D of $D1$ and $D2$ that subsumes C [Pla76].

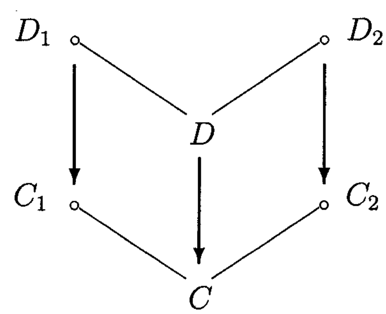


Figure 4.1: Subsumption theorem

Subsumed clauses can be deleted in most cases. For example, a resolvent can be deleted if it is subsumed by an input clause or a clause that was previously derived by resolution. Resolution produces a clause that subsumes any clause that can be derived by generating ground instances of clauses and carrying out resolutions on them.

In HERBY, the most general substitutions occur only with a few atom selection heuristics, which will be discussed later (Section 4.5). For example, suppose the following clauses are considered in the atom selection heuristics.

1.$P(a, x, y) \vee \neg C(y)$
2.$\neg P(x', b, y')$
3.$C(f(b))$

Then, two atoms are generated: $C(f(b))$, created by resolving clause 1 and clause 3 with the substitution $\{f(b)/y\}$ and $P(a, b, f(b))$, generated by resolving the resulting resolvent of clause 1 and clause 2 with the substitution $\{a/x', b/x, f(b)/y'\}$. They are the most general instances under the given clauses.

However, atoms grounded by the fixed grounding do not guarantee the most general substitutions in clauses. These atoms can generate clauses subsumed later by the most general instances. Consequently, they can prevent a semantic tree from closing or make it to find a longer proof. The goal of a grounding strategy is to find an atom that is general enough to prevent such circumstances.

4.1 Semantic trees with ungrounded atoms

With respect to grounding variables, we might think of a semantic tree with ungrounded atoms. In this case, a unit clause or a literal picked by atom selection heuristics will be used without applying a grounding strategy beforehand. The variables in the atom are grounded whenever needed during semantic tree construction. The semantic tree will eventually achieve the goal of having the most general atoms under the given clauses, thus eliminating the necessity to determine what variables have to be grounded with what constants. Therefore, such a semantic tree does not need the extra work incurred by having unifiers that are not the most general.

This form of semantic tree indeed exists. It is called a *semantic tree with ungrounded atoms* or a *semantic tree with variables.* It is a simple extension of the semantic trees we considered and was proposed by Plaisted as a mathematical object. In fact, there is a correspondence between a semantic tree with variables and a *resolution-refutation proof tree* [Pla76].

The resolution-refutation proof tree is a binary one in which every node is labelled with a clause. Each clause labelling the father node must be a resolvent of clauses labelling son nodes. This tree preserves the entire structure of the resolution proof, except that it includes a separate copy of a subtree for each occurrence of the same resolvents in the proof.

Consider theorem A in Figure 4.2. A resolution-refutation proof tree (Figure 4.3) and a corresponding semantic tree with ungrounded atoms (Figure 4.4) are shown.

1. $P(x) \vee \neg Q(x)$
2. $Q(a) \vee Q(b)$

negated_conclusion

3. $\neg P(x)$

Figure 4.2: Theorem A

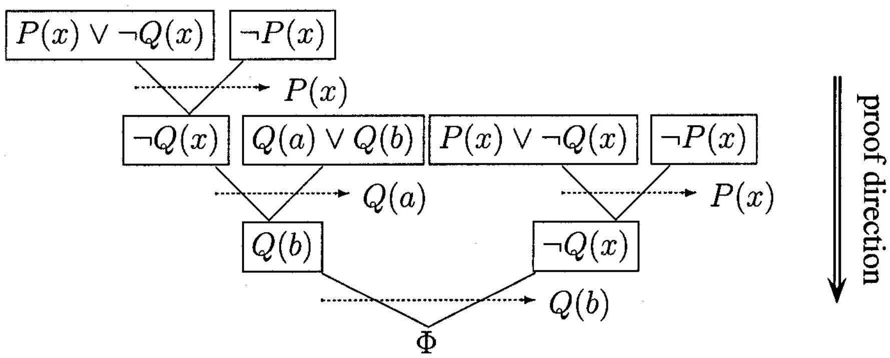


Figure 4.3: A resolution-refutation proof tree of the theorem A and atoms obtained

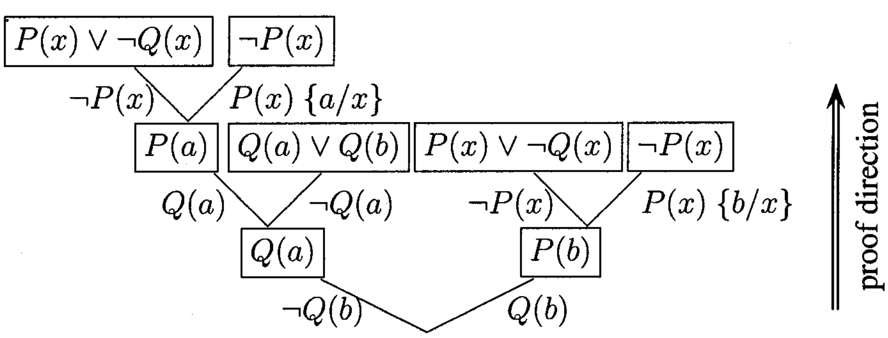


Figure 4.4: A corresponding semantic tree with the ungrounded atoms of Figure 4.3

Note that the semantic tree is drawn with root low and leaves high, and it is irregular where it contains different atoms at the same level. The leaves indicate the clauses that are inconsistent with atoms on the path.

A semantic tree with variables is basically a resolution-refutation proof tree in which the branches are labelled with instances of the literals resolved away in the resolution of each clause. These instances are chosen so that a clause C in S fails at a node N when C has a ground instance $C\sigma$ such that atoms from the root of the semantic tree to the node N logically imply $\neg C\sigma$. As indicated in Figure 4.3 by dashed lines, atoms are obtained by finding the most general instances of resolved away literals. If there are variables, these remain ungrounded. When the semantic tree is constructed, ungrounded atoms are grounded when necessary.

4.2 Difficulty using ungrounded atoms

Despite the correspondence mentioned above, using ungrounded atoms in constructing semantic trees is impractical. Note that the proof direction reverses that of the resolution-refutation proof tree. The grounding in a branch must be consistent with all other branches at the same level and with all branches at all levels under the current one that are not deployed yet. Otherwise, the proof will be incorrect. As an example, we show an incorrect proof of theorem S13ROB1 from the Stickel test set using a slightly modified HERBY for this purpose:

$1:\ p(x,y)$

$2:\ \neg p(y,f(x,y)) \vee \neg p(f(x,y),f(x,y)) \vee q(x,y)$

negated_conclusion

$3:\ \neg p(y,f(x,y)) \vee \neg p(f(x,y),f(x,y)) \vee \neg q(x,f(x,y)) \vee \neg q(f(x,y),f(x,y))$

Four more clauses are added by the BCR heuristic. Parentheses and disjunction symbols are omitted.

$4\ (3a, 1a)\ \neg pfxyfxy\ \neg qxfxy\ \neg qfxyfxy$

$5\ (3b, 1a)\ \neg pxfyx\ \neg qyfyx\ \neg qfyxfyx$

$6\ (2a, 1a)\ \neg pfxyfxy\ qxy$

$7\ (2b, 1a)\ \neg pxfyx\ qyx$

Figure 4.5 shows the semantic tree constructed using ungrounded atoms. The $\#n$ notation at the end of the literals indicates that the literal is resolved away with the n^{th} atom on the path to the root of the tree. For the first atom, in this case, let us suppose that pxy is chosen because it is the only unit clause available in the clause list. The first literal $\neg pxy$ closes the left branch. The other branch generates 6 more clauses. Clauses 8 and 9 are deleted, because they are subsumed by existing clauses and denoted by # at the end of the clause number.

$8\#\ (1a, 2a)\ \neg pxfyx\#1\ \neg pfyxfyx\ qyx$

$9\#\ (1a, 3a)\ \neg pxfyx\#1\ \neg pfyxfyx\ \neg qyfyx\ \neg qfyxfyx$

$10\ (1a, 4a)\ \neg pfxyfxy\#1\ \neg qxfxy\ \neg qfxyfxy$

$11\ (1a, 5a)\ \neg pxfyx\#1\ \neg qyfyx\ \neg qfyxfyx$

$12\ (1a, 6a)\ \neg pfxyfxy\#1\ qxy$

$13\ (1a, 7a)\ \neg pfxfxy\#1\ qyx$

Atoms 2 and 3 are chosen by the second atom selection heuristic(ASH2) as follows:

```
GENERATE ATOM USING ASH2:
C1,C2 resolve to C4, which resolves with C3
```

$C1 : 13\ (1a, 7a)\ \neg pfxfxy\#1\ qyx$

$C2 : 11\ (1a, 5a)\ \neg pxfyx\#1\ \neg qyfyx\ \neg qfyxfyx$

$C3 : 12\ (1a, 6a)\ \neg pfxyfxy\#1\ qxy$

$C4 : 14\ (13a, 5a)\ \neg qfxyfxy$

```
The atom generated is : 15: qxfxy
   2: qfaafaa
   3: qafaa
```

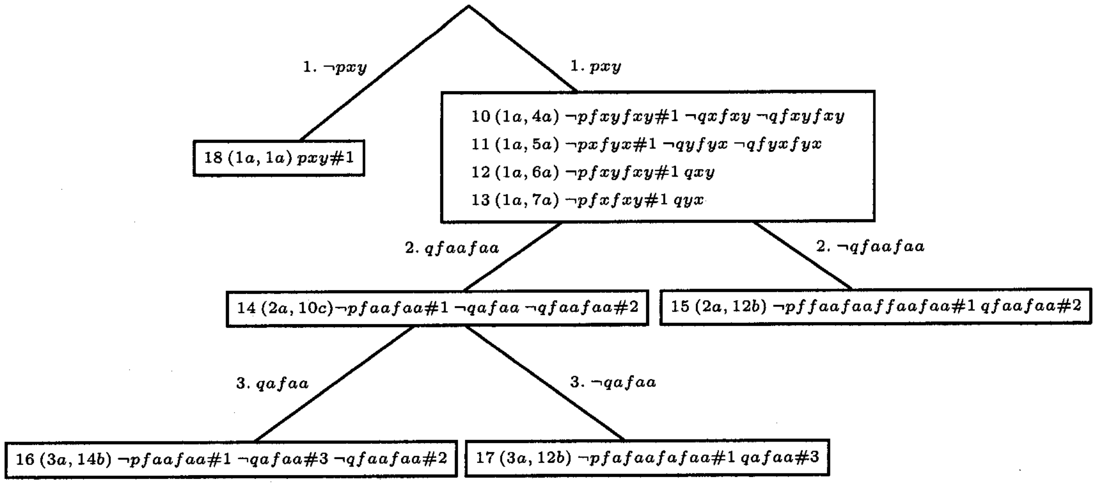


Figure 4.5: S13ROB1 : An incorrect semantic tree with ungrounded atoms

Atoms 2 and 3 are grounded in this example for an illustration. As there are no constants in this theorem, grounding them with an arbitrary constant a does not affect the discussion.

In this example, the literals resolved away with the first atom pxy are unified differently when the atom 2 and 3 resolved with the other literals in the clauses.

16 $(3a, 14b)$ $\underline{\neg pfaafaa}\#1$ $\neg qafaa\#3$ $\neg qfaafaa\#2$

17 $(3a, 12b)$ $\underline{\neg pfafaafafaa}\#1$ $qafaa\#3$

The first atom pxy resolves away the two same literals of clauses 10 $\neg pfxyfxy\#1$ and 12 $\neg pfxyfxy\#1$ at the beginning. It turns out, however, that they are grounded differently depending on the remaining literals as the tree goes deeper.

We may think that the already resolved literals do not affect the semantic tree construction. This is not true, however. Observe that in Figures 4.3 and 4.4, resolution tries to find a Φ among the given clauses from the bottom up. Only two clauses are involved in a unification of resolution. The semantic tree on the other hand, attempts to close the tree from the top down. The atoms therefore, must satisfy unifications of all unifiable clauses in the subtree. In this example, however, the unification later exposes an inconsistency.

The general rule of the selection of an ungrounded atom is not to choose an atom

```
Chained_resolution(Atom A)
{
    if (A does not contain any variables) return A
    Let R ← A
    for (all clauses C in clause list) {
        for (all literals L in C) {
            If (∃μ s.t. Rμ ≡ Lμ or ¬Rμ ≡ Lμ) {
                R ← Rμ
                if (R does not contain any variable) return R }}}
    return fixed grounding
}
```

Figure 4.6: Algorithm of chained-resolution grounding

that subsumes a given clause. As the definition of the subsumption says (2.1 Preliminaries), if an ungrounded atom subsumes a given clause, there is a likelihood that the remaining literals of the subsumed clause are instantiated differently as the tree goes deeper. In the above example, pxy is not an appropriate atom. It simultaneously subsumed the given clauses 2 and 3, but used to resolve two literals, 10 $\neg pfxyfxy\#1$ and 12 $\neg pfxyfxy\#1$. Later, these are instantiated differently according to the remaining literals.

4.3 Chained-resolution grounding

To determine whether one clause subsumes another is a computationally intensive procedure. Moreover, it is not clear how to apply a consistent grounding scheme in a semantic tree with ungrounded atoms, although it helps to comprehend the relation with the resolution proof.

In this thesis, semantic trees use a newly proposed grounding scheme. The following is relevant to the scheme. *Chained-resolution grounding* [Lap98] described in Figure 4.6 is a scheme using a series of resolutions with the literals on the path to the root of a tree to obtain ground instances as in Figure 4.6. Let A be the atom we are grounding. The scheme ensures that either A or $\neg A$ resolves with clauses on the path to the root of the tree.

We give partial proof of theorem S19APABH from the Stickel test set as an example.

1. $\neg a(x, e, y) \vee \neg a(x, t, y)$
2. $\neg i(m(x), d(l, y))$
3. $\neg r(h)$
4. $a(h, z, d(g(z), y))$
5. $a(m(s), e, n)$
6. $\neg a(m(x), z, d(g(z), y)) \vee a(m(x), z, y) \vee i(m(x), y)$
7. $\neg a(m(x), z, y) \vee \neg a(h, z, y) \vee i(m(k(y)), d(p, y))$
8. $\neg a(h, z, y) \vee a(m(x), z, y) \vee \neg i(m(x), y)$
9. $\neg a(m(x), z, y) \vee a(h, z, y) \vee \neg i(m(x), y)$
10. $\neg a(x, t, y) \vee c(y) \vee \neg r(x)$
11. $\neg a(m(x), z, y) \vee a(m(x), z, d(g(z), y))$
12. $\neg i(m(x), y) \vee i(m(x), d(g(z), y))$
13. $\neg a(h, z, y) \vee a(m(k(y)), z, y)$
14. $\neg a(x, z, y) \vee a(x, z, d(p, y))$
15. $\neg a(x, z, y) \vee a(x, z, d(l, y))$
16. $\neg a(x, z, d(l, y)) \vee a(x, z, y)$
17. $\neg a(x, e, n) \vee r(x)$

$negated\_conclusion$

18. $\neg c(y)$

The semantic trees illustrating the output generated by HERBY are given in Figure 4.7. Each node is identified by numbers inside circles, and each path is identified by its numeric label. Number 1 identifies a left branch and number 2 a right branch.

At the output of `S19APABH.THM`, the notation $(1a, 2b)$ means that the first literal of the chosen atom is resolved with the second literal of clause 2. The # symbol after a literal indicates the literal is resolved away. The same symbol after a clause number indicates that the clause is deleted somehow.

The first atom comes from the first literal of clause 17 and contains a variable x. $\neg axen$ is chosen by heuristic 5 (ASH5a) explained in section 4.6. The heuristic searches for a unit clause with one variable, at most, among expanded clauses by the BCR heuristic, which is not the case here. Next, starting from the last clause, it searches for a clause with two literals and one variable at most.

At node 1, the atom $axen$ is resolved with the ground[1] atom $\neg ahen$, thus substituting h for x. At this point, the atom is replaced with $ahen$ and tree construction is repeated.

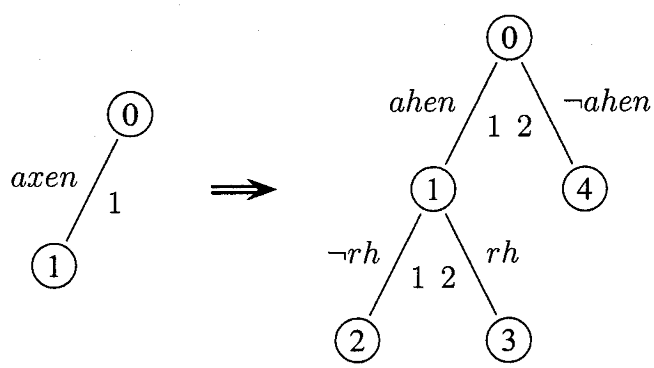


Figure 4.7: Partial semantic trees for the output of S19APABH

```
Theorem: S19APABH.THM

Predicates: a i r c
Functions:  e t p n l h s : g d k m
ESAF:
ESAP:

NEXTC=18 TIME=3600 XARS=35

GENERATE ATOM   1: ~axen    H5a.17   T0   N1   C0   U12
```

[1] e, t, p, n, l, h and s are constants and g, d, k and m are functions

```
Branch on atom:    1: axen to node 1

GENERATE CLAUSES AT NODE 1
 19: (1a,1a)    ~axen#1  ~axtn
 20# (1a,7a)    ~amxen#1  ~ahen imkndpn
 21# (1a,7b)    ~amxen ~ahen#1  imkndpn
 22: (1a,8a)    ~ahen#1  amxen ~imxn
 23: (1a,9a)    ~amxen#1  ahen ~imxn
 24: (1a,11a)   ~amxen#1  amxedgen
 25: (1a,13a)   ~ahen#1  amknen
 26: (1a,14a)   ~axen#1  axedpn
 27: (1a,15a)   ~axen#1  axedln
 28: (1a,17a)   ~axen#1  rx
 29: (1a,7b)    ~amxen#1  ~ahen#1  imkndpn
GENERATED 11 CLAUSES

   .
   .
   .

Branch on atom:    1: ahen to node 1

GENERATE CLAUSES AT NODE 1
 19: (1a,1a)    ~ahen#1  ~ahtn
 20: (1a,7b)    ~amxen ~ahen#1  imkndpn
 21: (1a,8a)    ~ahen#1  amxen ~imxn
 22: (1a,13a)   ~ahen#1  amknen
 23: (1a,14a)   ~ahen#1  ahedpn
 24: (1a,15a)   ~ahen#1  ahedln
 25: (1a,17a)   ~ahen#1  rh
GENERATED 7 CLAUSES

PATH: 1

GENERATE ATOM USING ASH1:
C1 and C2 resolve to the NULL clause

C1:   25: (1a,17a)  ~ahen#1  rh
C2:    3: ~rh
The atom generated is: 26: rh

  2: rh   H1  T19  N2  C7  U0

Branch on atom:   2: ~rh to node 2

GENERATE CLAUSES AT NODE 2

PATH: 11 FAILS:  27: (2a,17b)  ~ahen#1  rh#2

Branch on atom:   2: rh to node 3

GENERATE CLAUSES AT NODE 3

PATH: 12 FAILS:  26: (2a,3a)   ~rh#2
```

```
PATH: 1 FAILS:
Branch on atom:    1: ~ahen to node 4

GENERATE CLAUSES AT NODE 4
 19: (1a,9b)    ~amxen ahen#1   ~imxn
 20: (1a,16b)   ~ahedln ahen#1

  .
  .
  .
```

4.4 Grounding strategy of parallel chained-resolution

In this section, we propose a relatively simple grounding method that inherits the idea of the chained-resolution grounding, but utilizes the parallel environment thus, we believe, effectively captures the semantics of the given clauses.

The fixed grounding strategy is definitely context insensitive and naive. On the other hand, the chained-resolution grounding introduced in Figure 4.6 tries to deduce a ground atom naturally by finding a literal resolvable with a clause on the clause list. One disadvantage of the scheme is that it can be expensive: $\mathcal{O}(n)$ unification checks, where n is the number of literals on the active path. In addition, the scheme tends to be left or right subtree oriented, depending on the order of unification, since it takes the first ground literal as the next atom.

The algorithm can be modified to relieve such a drawback. For example, it can take the shortest ground literal from the unifications of both branches, with the cost of the increasing time complexity.

In this scheme, clause ordering is so important. When the clause list contains non-unifiable literals, the chained-resolution grounding is not different from the idea of the fixed grounding. We usually select clauses in the negated conclusion of a theorem first, followed by all the remaining clauses in reverse.

If we assume that maximizing the number of resolutions facilitates success, we can take the atom that maximizes the number of resolutions on a given path. In practice, however, this does not necessarily lead to success. Moreover, the time complexity required to find the atom is a major drawback.

Parallel chained-resolution is the solution to the problems just described. This is similar to chained-resolution grounding, except that each processor takes a different instance of the same atom. The procedure is given in Figure 4.8.

Example 9 In the previous theorem S19APABH, let us suppose an atom $axen$ is chosen and distributed to each slave. The first slave will have the first ground atom $ahen$ as a result of the following round of the parallel chained-resolution.

```
Branch on atom:   1:   axen to node 1
```

GENERATE CLAUSES AT NODE 1

19 :	$(1a, 1a)$	$\neg axen\#1$	$\neg axtn$	
20#	$(1a, 7a)$	$\neg amxen\#1$	$\neg ahen$	$imkndpn$
21#	$(1a, 7b)$	$\neg amxen$	$\neg ahen\#1$	$imkndpn$
22 :	$(1a, 8a)$	$\boxed{\neg ahen\#1}$	$amxen$	$\neg imxn$
23 :	$(1a, 9a)$	$\neg amxen\#1$	$ahen$	$\neg imxn$
24 :	$(1a, 11a)$	$\neg amxen\#1$	$amxedgen$	
25 :	$(1a, 13a)$	$\neg ahen\#1$	$amknen$	
26 :	$(1a, 14a)$	$\neg axen\#1$	$axedpn$	
27 :	$(1a, 15a)$	$\neg axen\#1$	$axedln$	
28 :	$(1a, 17a)$	$\neg axen\#1$	rx	
29 :	$(1a, 7b)$	$\neg amxen\#1$	$\neg ahen\#1$	$imkndpn$

The second and third slaves will have the same second and third ground atoms $ahen$ as shown in the dashed framed box. As for the fourth slave, the negation of the given atom is applied, and the $amsen$ is used instead of the given atom $axen$. If all the above attempts fail, fixed grounding is used as a last resort. An experiment with parallel chained-resolution strategy is presented in table 6.3.

Branch on atom: 1: $\neg axen$ to node 1

GENERATE CLAUSES AT NODE 1

19 : $(1a, 5a)$ $\boxed{amsen\#1}$

```
ground_atom(atom C)
{
    if (the atom selected does not contain variables) then return
    ground_instance = 0
    for (all clause C_i in clause list) {
        generate binary resolvents of (C, C_i) and (¬C, C_i);
        for (each resolvent R just generated) {
            for (each literal L in R ) {
                find a literal just resolved away
                if (the literal has no variable) {
                    ground_instance++
                    if (ground_instance == s) {
                        negate the literal /*to have the same sign of the atom*/
                        install the literal as the atom
                        return}}}}}
    if (all the above attempts fail)
        use fixed grounding
}
```

Figure 4.8: Grounding algorithm of parallel chained-resolution at slave s

4.5 Modified atom selection heuristics

In this section, we describe the heuristics used by PrHERBY except for the ASH_Parallel heuristic, which will be discussed in Figure 5.9.

These heuristics enable the system to choose atoms that will close the semantic tree at the nodes at which they are successful or likely will lead to the closure. The strategies used in ASH1 to ASH3 borrow the concept of *hyperresolution* where *electrons* consist only of atoms that are resolvable with the *nucleus* [CL73].

Example 10 Consider the following theorem:

1 Q

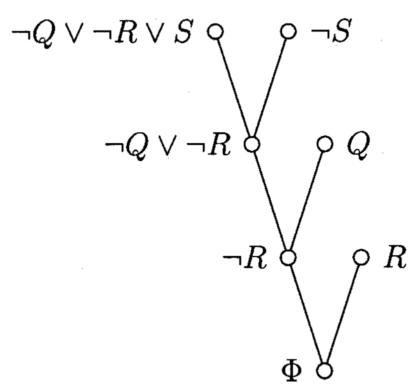
2 R

3 $\neg Q \vee \neg R \vee S$

Negated_conclusion

4 $\neg S$

If a hyperresolution succeeds in deriving a contradiction, as in the tree below, ASH3 obtains atoms S, Q and R by taking the most generally unified resolvent at each level of resolution. Then, the branch of the semantic tree where the atoms are obtained is closed.



UR (unit resulting) resolution is an inference rule that produces a unit clause (UR resolvent) from a set of clauses, one of which is a non-unit clause (*nucleus*) and the rest are unit clauses (*electrons*). ASH4 maintains the unit clauses resulting from the UR resolution and tries to find a contradiction among them. Most atom selection heuristics are those of HERBY with minor changes [New01].

Atom selection heuristic 1 (ASH1): Search for a pair of unit clauses that are resolvable to yield the Φ clause. If a pair is found, use the atom substituted with the *mgu* (most general unifier) of the two clauses as the next Herbrand base atom. The next two nodes both fail.

Atom selection heuristic 2 (ASH2): Search the clause list for three clauses $C1$, $C2$ and $C3$, in which $C1$ and $C2$ are unit clauses and $C3$ has two literals. $C1$ resolves with the first literal of $C3$ to yield a clause $C4$, which in turn resolves with $C2$ to yield the Φ clause. If a trio is found, two unit clauses substituted with *mgu*s are used as the next Herbrand base atoms. After proceeding to the next two nodes, the branches are closed.

Atom selection heuristic 3 (ASH3): Search the clause list for four clauses $C1$, $C2$, $C3$ and $C4$. $C1$, $C2$ and $C3$ are unit clauses, and $C4$ has three literals. $C1$ resolves with the first literal of $C4$ to yield a clause $C5$. That, in turn, resolves with $C2$ to yield a clause $C6$. And that, in turn, resolves with $C3$ to yield the Φ clause. If a quad is found, three unit clauses substituted with *mgu*s are used as the next Herbrand base atoms. After proceeding to the next three nodes, the branches are closed.

Atom selection heuristic 4 (ASH4): A unit clause list is maintained by resolving a unit clause with clauses containing two literals, or two other unit clauses with clauses containing three literals. Then, if any two unit clauses on the list resolve to yield the Φ clause, a unit clause substituted with the *mgu* is chosen as the next atom. This heuristic does not guarantee a node failure, since the unit clauses might come from different paths in the tree. Otherwise, the node in which the atom is found will fail.

Atom selection heuristic 5 (ASH5): Search the base clauses for a unit clause that has not been previously selected, and select it for the next atom. If it is not available, search the remaining clause list for a unit clause. One child of every node at which this heuristic chooses an atom will be a failure node.

4.6 Atom guiding heuristics

The heuristics, ASH1 to ASH3, create a closed semantic tree at the nodes at which they are successful. These heuristics are greedy in the sense that they work locally but do not guarantee overall success. However, by trying to close each branch, the atoms guide the proof so that the whole tree eventually closes.

The ASH_Parallel heuristic discussed later (Figure 5.9) delivers atoms that are expected to accomplish the role of ASH1 to ASH3. Besides these, each slave depends on the other heuristics. According to experiments, ASH5 is used most often. ASH5 is called after the ASH_Parallel heuristic.

In the use of ASH5, if a clause set contains

$$Equal(a, a)$$

$$\neg Equal(x, y) \vee Equal(g(x), g(y))$$

which correspond to a subset of S26WOS2 in the Stickel test set, it is possible to generate unit clauses repeatedly if ASH5 is not used carefully. This happens when $Equal(a, a)$ resolves with $\neg Equal(x, y)$ and generates

$$Equal(g(a), g(a))$$

ASH5 next chooses $Equal(g(a), g(a))$ and generates

$$Equal(g(g(a)), g(g(a)))\ldots$$

To handle this problem, we refine and replace ASH5 as follows:

Atom selection heuristic 5a (ASH5a)

- Choose a ground unit clause with an opposite sign from the previously selected atom in the clause list excluding base clauses.
- Choose a ground unit clause with the same sign as the previously selected atom in the clause list excluding base clauses.

- Choose a unit clause with one variable in the clause list excluding base clauses.
- Choose a literal from a clause containing two literals and one variable, at most, whenever possible in the clause list.

Atom selection heuristic 5b (ASH5b)

- Choose a ground literal from the unit clause list collected by ASH4 with an opposite sign from the previously selected atom.
- Choose a ground literal from the unit clause list collected by ASH4 with the same sign as the previously selected atom.
- Choose a literal from the unit clause list with one variable.
- Choose a literal from the unit clause list with more than one variable.

Atom selection heuristic 5c (ASH5c)

- Search the base clauses and choose a literal with at most one variable in the clause list.
- Search the clause list excluding base clauses and choose a literal with at most one variable in the clause list.
- Choose a literal of a clause in the clause list.

4.7 Unit-list-passing heuristic

ASH4 maintains a unit list containing unit clauses found at nodes where atoms are selected by resolving a unit clause with another clause containing two or three literals. If two unit clauses on the list resolve with the *mgu* μ to yield the Φ clause, then the unit clause after applying μ is selected as the next atom. This atom is preferred over the one selected by ASH5. The atom does not guarantee the closure of the node where the atom is found because the two unit clauses may not be on the same path to the root of the tree. Nonetheless, ASH4 has been found to be a powerful heuristic and is often the key to proving difficult theorems [New01].

In this parallel implementation, each slave also maintains the unit list. In addition to that, the *unit-list-passing heuristic* combines the unit lists in an attempt to find useful atoms. The master initiates the transfer at the beginning. At intervals, slave $i - 1$ passes its unit list to slave i. After receiving the unit list, each unit clause in the list is examined to see if it is resolvable with a unit clause in the unit list of slave i. If it is, then the clause is selected as the next atom; otherwise, the clause is added to the unit list of slave i.

Unfortunately, it takes time to process the whole unit list with the current, simple comparison method. Therefore, the configuration for the experiment uses this heuristic only once for each slave during the time limitation.

Despite the fact that the chances of finding resolvable pair in the list are slim as we attempt to exploit various search spaces by generating atoms in quite different orders, the potential of the unit-list-passing heuristic lies in the fact that semantic trees tend to be thin. It increases the chance of successful selection of atoms as the size of the unit list grows larger.

The success of this strategy relies on the linear property of semantic trees and the search technique. The effectiveness of this strategy is never less than ASH4. In fact, it is a parallel extension of the ASH4.

Chapter 5

Exploiting Parallelism: Highly Competitive Computing Model

5.1 Parallel systems

With the rapid development of computer technology and algorithmic approaches that help to speed up inferences, it is very likely that the performance of provers will improve. However, the search space of the most interesting theorems still remains enormous. Due to the emergence and growth of parallel architectures, parallelization is considered to have great potential to attack the problem. Several attempts have been made to parallelize automated theorem proving systems. A taxonomy of parallel strategies for deduction, by Maria Paola Bonacia and Jieh Hsiang [BH94], surveys them thoroughly. Here, we classify parallel systems into Prolog based, resolution based, and semantic tree based systems.

Prolog technology theorem prover [Sti88] is basically an extension of Prolog's inference mechanism to first-order logic, based on the model elimination principle [Lov69b]. Well-known parallel Prolog technology theorem provers include PARTHENON [BCLM92], PARTHEO [SL90] and METEOR [Ast94]. In these provers, each concurrent process has access to the input set of clauses and tries to apply a clause to one of the current goals. Each process selects an input clause, possibly a different one for each process, and tries to resolve it with the input goal generating new subgoals. The distribution of tasks is done by task stealing in which a process obtains new tasks from the queues of other processes.

Another branch in Prolog technology theorem prover is SETHEO [Let92] based systems. The main proof unit is based on a refinement of the connection method, which is, in effect, identical to the model elimination procedure. SETHEO itself is not a parallel system, but many parallel systems are based on it. Examples are SiCoTHEO [Sch97], SPTHEO [Sut99], RCTHEO [Ert92] and CPTHEO [FW98].

PARROT [JO92] is a **resolution based**, parallel version of OTTER. OCTOPUS [New98] is also a parallel version of THEO. PARROT introduces parallelism by allowing multiple processes to generate resolvents. The parallel deduction system based on this scheme consists of one master process and several slaves. The master selects clauses for each slave, and each slave generates resolvents from them.

OCTOPUS, too, runs with one master and as many slaves as are available. Each slave carries out the same search procedure, but each sets its own limits on the number of literals in a clause and the number of constants, functions, and variables in a literal. Some processors use the set-of-support strategy in addition.

PHERBY [AN98] is a parallel version of the **semantic tree based** prover HERBY. It exploits parallelism by using a larger set of heuristics than HERBY and uses cooperative computing to determine atom quality.

Besides the classification of Prolog based, resolution based and semantic tree based systems, we can also classify parallel theorem provers into two categories: one divides the search space among several slaves, which necessarily increases communication overhead and requires strategies for fair task distribution. The other runs multiple copies of the serial theorem prover with different settings.

The parallel versions of THEO and HERBY, OCTOPUS and PHERBY, are competitive models running the same slaves with different settings. In the case of THEO, dividing work among slaves is not practical in a loosely coupled distributed environment due to dependency on a large hash table. It is not clear how to join hash tables distributed in each processor and find a contradiction. In the case of semantic tree, the work can be divided among slaves, since semantic tree branches do not depend on each other. One problem, however, is synchronizing the atoms at each level. If we ignore synchronization and use irregular semantic trees instead, fair task distribution is still a difficulty, because semantic trees as found by HERBY tend to be thin. A goal is made to generate a scheme which is scalable to a large number of processors.

Using various strategies to prove theorems has proved useful. Gandalf, developed by Tanel Tammet, uses a time slicing method for each strategy and defeated competing provers in CADE 1997 and 1998. Although Gandalf is a serial prover, the slicing concept fits well in parallel systems. Each processor automatically chooses a different set of strategies, ones that are probably appropriate for the given theorem. It requires little communication among processors and is easy to implement. Its major disadvantage is that the number of different, relevant strategies is limited and there is much effort overlap among competing strategies. The system is not scalable, therefore, and returns diminish rapidly as the number of processors increases.

In this chapter, we present a scheme to achieve a large freedom of scalability— that is, generally producing increasing returns as the number of processors is increased. We introduce a resolution method used in the master of the parallel scheme in the next section.

5.2 IDDFS resolution

IDDFS(Iterative Deepening Depth-First Search) is an algorithm that suffers the drawbacks of neither breadth-first nor depth-first search on trees. It first performs a depth-first search to depth one. It then discards the nodes generated so far, starts over, and performs a depth-first search to depth two. Once again, it starts over and performs a

```
IDDFS()
{
while(iter_depth < MAX_ITERATIONS) {
    iter_depth++;            /* increase iteration depth */
    search_tree();           /* Begin search */
}
search_tree()
{
if (N is the root node ) {
    for (all base clauses C_i) {
        generate factors of C_i
        for (j = i - 1; j >= 1; j - -)
            generate binary resolvents of (C_i, C_j)}}
else {
    for (all base clauses C_i) {
        generate resolvents of C_i and inference I_d}
        generate factors of I_d
        for (all inferences I_i) {
            generate resolvents of I_i and with inference I_d}}
}
```

Figure 5.1: IDDFS algorithm for a theorem with clauses C_i at some node N at depth D and with inferences I_i on the path to node N

search to depth three, continuing this process until a goal state is reached. This algorithm is guaranteed to find the shortest-length solution with a minimal use of space. The disadvantage of IDDFS is that it performs extra computations before reaching the goal. Nonetheless, it has been shown that this wasted computation does not affect the asymptotic growth of the run time of exponential tree searches [Kor85].

When THEO tries to find a proof, it carries out an IDDFS. The root of the tree corresponds to a set of base clauses. Branches leading from one node to another correspond to inferences that can be performed on the clauses at which the branches are rooted. Each node other than the root consists of clauses generated by the inference on the branch leading to it. On the first iteration, a search for a linear proof of length one is performed. On the second, a search for a proof of length two is carried out, and so on.

Although the semantic tree method has a binary branching factor, resolution has a much larger branching factor (on the order of b^d where b is the number of clauses and d is the depth of the node), because each node is generated by resolving together all pairs of base clauses, factoring individual base clauses, and then resolving each of the resulting clauses with some of the base clauses. Resolution strategy in theorem proving generally results in very short, fat trees. We describe the procedure in Figure 5.1.

5.2.1 An example

Consider theorem A in Figure 4.2. Figures 5.2 and 5.3 show the iterations that IDDFS carries out on the theorem. On the first iteration when the depth is one, three clauses are generated and no proof is found. On the second iteration, six more clauses are generated and no proof is found. On third iteration, while expanding the fourth clause, the Φ clause is finally generated.

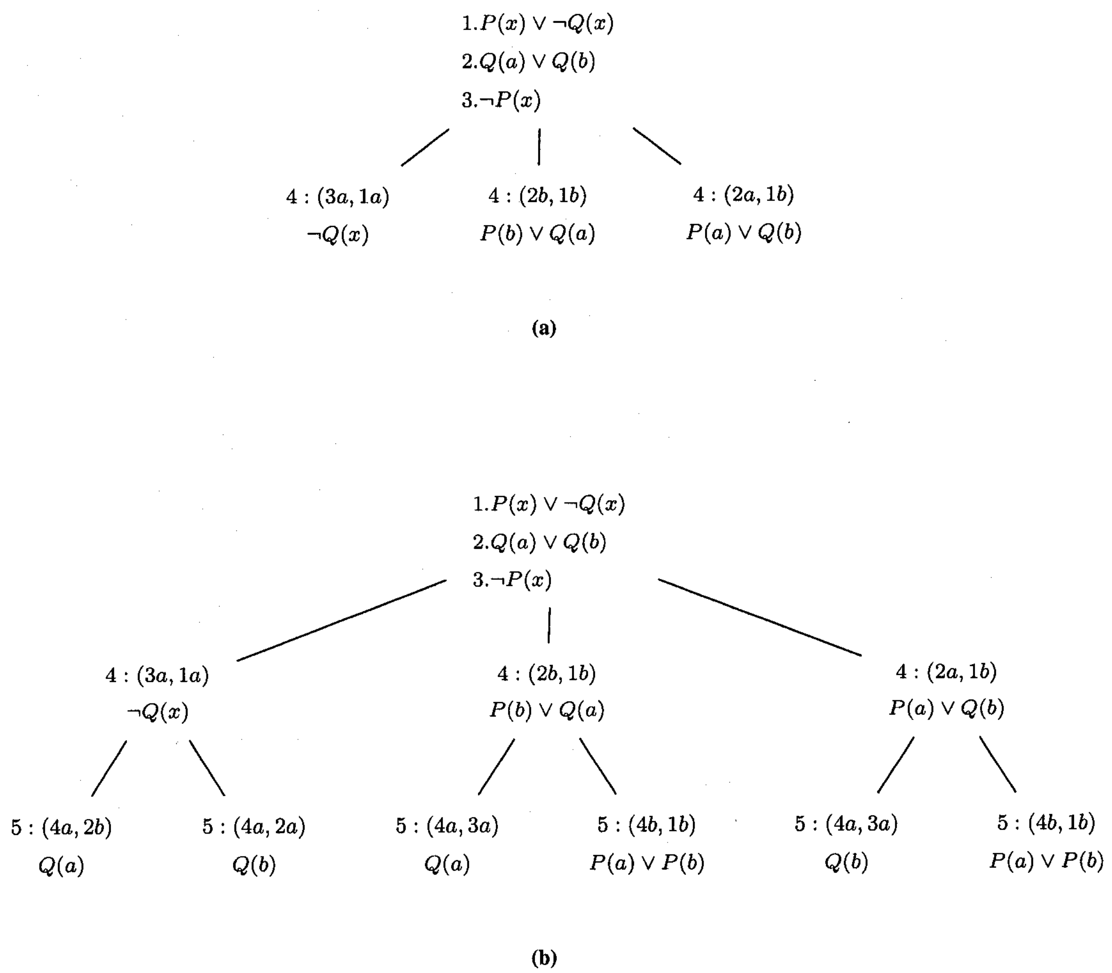
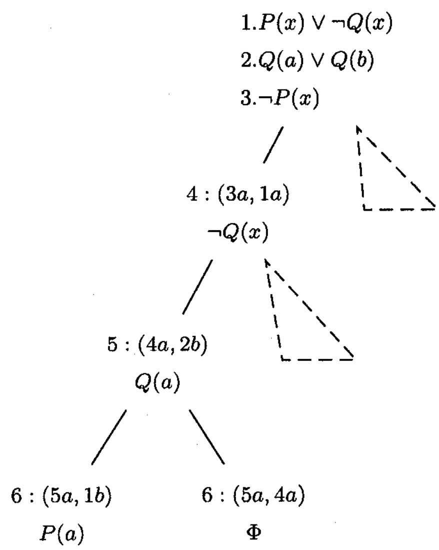


Figure 5.2: **(a)** first and **(b)** second iteration of theorem A



(c)

Figure 5.3: **(c)** third iteration of theorem A

5.2.2 Search strategies

Several search strategies can be used during IDDFS to restrict the number of generated clauses. In some cases, although these strategies can result in longer proofs, they usually do reduce the search space and yield proofs in less time.

Merge proof search generates binary resolutions at each node in the tree with the clauses at that node and with a parent node that contains a merge clause. If merge clauses are not available, the resolution happens only with a base clause in the root node.

NC (Negated Conclusion) search requires that inferences at the first level must include one clause from negated conclusions.

Extended search is carried out on level n of a tree on the n^{th} iteration if a clause C at level n or deeper can be resolved with a clause C' that has only one literal and that is an ancestor of C. The extended search strategy is an attempt to pursue clauses that are more likely to lead to a contradiction. Therefore, if a theorem has a proof of length n, a proof can be found at an iteration k, where $k \leq n$. In Figure 5.3, the extended search strategy finds a proof without performing the next iteration.

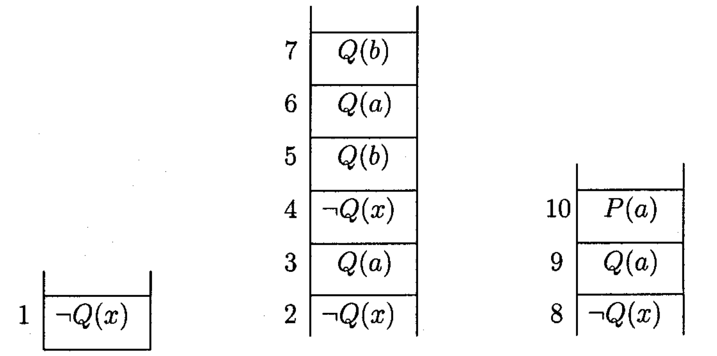
IDDFS generates clauses continually until it finds a proof. By definition, it gets increasingly closer to a proof. When it applies to resolution, for example, the extended search strategy tries to take advantage of this property; it examines unit clauses placed deeper in some circumstances to see if the contradiction is reached. In a semantic tree construction, however, choosing appropriate atoms—that is, ones that close the tree or lead to the tree closure—is the most important factor. We are concerned mainly, therefore, with the development of these atom-selection algorithms.

Unfortunately, choosing appropriate atoms before the construction of a semantic tree is a time consuming task and they are not easily found in the most interesting theorems. In fact, we experienced that picking literals randomly among clauses was ineffective. In HERBY, the random selection is the last resort (`ASH5c` in section 4.6). We discuss the use of IDDFS resolution in the next section.

5.3 Parallel semantic tree generation

The concept that IDDFS gets closer to a proof as it deepens brings an interesting idea that the search space can be explored by constructing semantic trees simultaneously. We distribute the atoms collected by the master among the processors so that each can construct its own distinct semantic tree. Each processor will construct a somewhat different semantic tree, depending on the atoms given to it by the master. The semantic tree will be different according to the behavior of the IDDFS execution and the number of unit clauses generated. Even the same atoms repeatedly collected by different iterations will be distributed in a different order in a very systematic way.

As an example, consider an atom distribution scenario. The atoms collected in Figures 5.2 and 5.3 are shown in Figure 5.4. Atoms are numbered sequentially according



(a) 1^{st} *iteration* (b) 2^{nd} *iteration* (c) 3^{rd} *iteration*

Figure 5.4: Atoms collected at each iteration.

to the order of generation by IDDFS.

Starting from the first atom in the Figure 5.4 (a), each is distributed to each slave. In this scheme, slave i receives the i^{th} atom which is **modulo** $number\ of\ slaves$. The first slave receives the atom $\neg Q(x)$ and constructs a closed semantic tree with atoms:

1. $\neg Q(x)$ from the master. Applying parallel chained-resolution grounding, we obtain

   1. $P(x) \vee \neg Q(x)$
   2. $\boxed{\text{Q(a)}} \vee Q(b)$
   3. $\neg P(x)$

   The first slave takes the first ground atom $\neg Q(a)$ and generates a resolvent (1a,2a) $Q(b)$.

2. $Q(b)$ from atom selection heuristics, because the branch is closed by resolving the (1a,2a) $Q(b)$ with clause 1 and resolving the resolvent $P(b)$ with clause 3. The selected atom $Q(b)$ then generates a resolvent (2a,1b) $P(b)$.

3. $P(b)$ from atom selection heuristics.

4. $P(a)$ from atom selection heuristics. The semantic tree is shown in Figure 5.5.

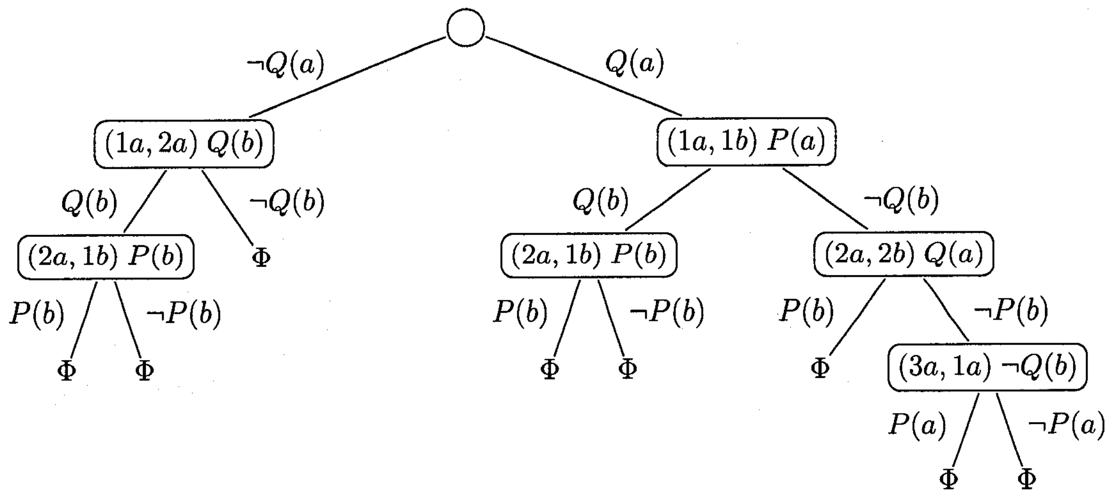


Figure 5.5: Semantic tree construction at the first slave

As indicated in Figure 5.4, the second iteration begins generating $\neg Q(x)$. The second slave receives the same atom $\neg Q(x)$ as the first slave. However, it is grounded differently.

1. $\neg Q(x)$ from the master. Applying parallel chained-resolution grounding, we obtain:

   1.$P(x) \vee \neg Q(x)$
   2.$\boxed{Q(a)} \vee \boxed{\text{Q(b)}}$
   3.$\neg P(x)$

   The second slave takes the second ground atom $\neg Q(b)$ and generates a resolvent (1a,2b) $Q(a)$.

2. $Q(a)$ from atom selection heuristics because the branch is closed by resolving (1a,2b) $Q(a)$ with clause 1 and resolving the resolvent $P(a)$ with clause 3. The selected atom $Q(a)$ then generates a resolvent (2a,1b) $P(a)$.

3. $P(a)$ from atom selection heuristics.

4. One more atom $P(b)$ is necessary to close the whole tree. The closed semantic tree is shown in Figure 5.6.

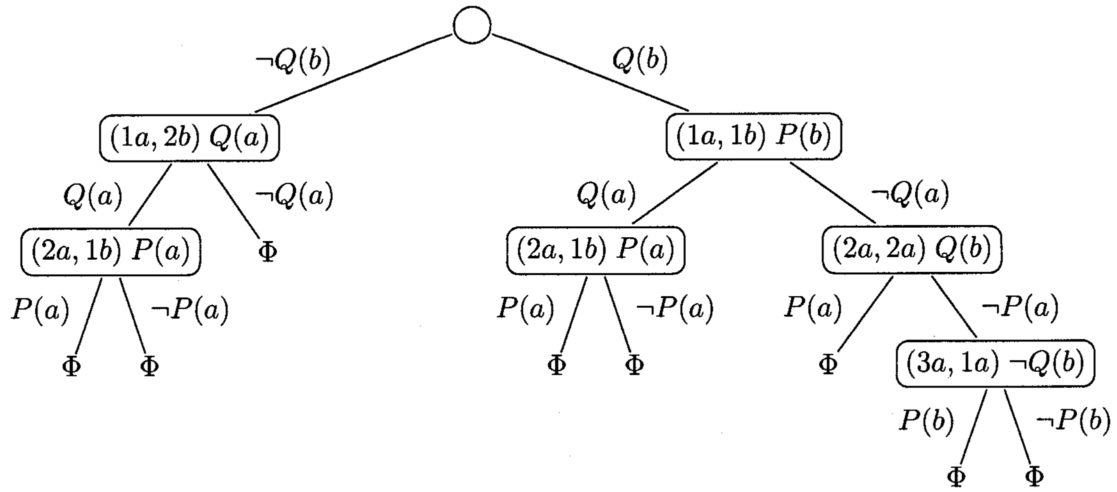


Figure 5.6: Semantic tree construction at the second slave

The third slave with the atom $Q(a)$ constructs a closed semantic tree very similar to the first slave except for the order of the selected atoms.

The fourth atom $\neg Q(x)$ is the same as the first and second one. Because the grounding algorithm cannot find the fourth ground atom in the given theorem, $\neg Q(a)$ is used by the fixed grounding strategy, and the fourth slave acts like the first.

The fifth to ninth slaves repeat the procedure described above with atoms 5 $Q(b)$, 6 $Q(a)$, 7 $Q(b)$, 8 $\neg Q(x)$, 9 $Q(a)$ if slaves are available. Otherwise, the redundant atoms are discarded.

With the tenth atom $P(a)$, the slave will construct a semantic tree with atoms:

1. $P(a)$ from the master. It closes a branch resolving with clause 3. The negated atom $\neg P(a)$ generates a resolvent (1a,1a) $\neg Q(a)$.

2. $\neg Q(a)$ from atom selection heuristics because the branch is closed by resolving (1a,1a) $\neg Q(a)$ with clause 2 and resolving the resolvent $Q(b)$ with clause 1, and resolving the resolvent $P(b)$ with clause 3.

3. $Q(b)$ from atom selection heuristics.

4. $P(b)$ from atom selection heuristics. The semantic tree is shown Figure 5.7.

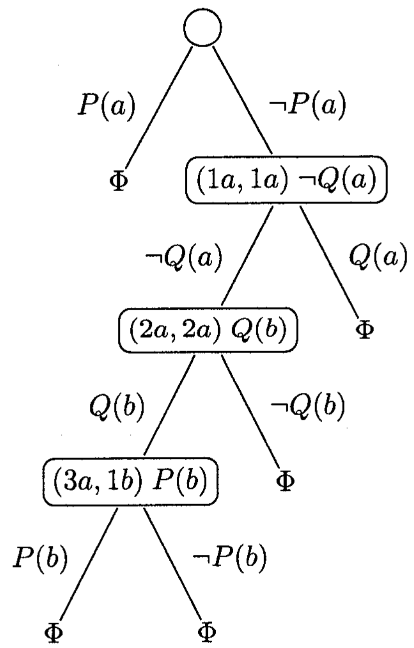


Figure 5.7: Semantic tree construction at the tenth slave

Note that atom $P(a)$ from the master is hidden inside the set of clauses and difficult to locate by means of atom selection heuristics without depending on luck. The scheme we propose provides the opportunity to generate these atoms.

The scheme will inevitably generate previously generated atoms repeatedly. However, if the extended search strategy is used to collect atoms, only the atoms collected in Figure 5.4 (a) and (c) will be used in this case. Our implementation permits the number of redundancies to be proportional to the number of processors. If the previously generated atom contains variables, this can have the effect of generating different, useful atoms due to the parallel chained-resolution grounding strategy. Also, the atoms are delivered to processors at different times making them to construct different semantic trees. Redundancy is an important factor, moreover, in achieving the algorithm's scalability.

The pseudo-codes for generating atoms from the master and for receiving atoms at a slave are given in Figures 5.8 and 5.9. The **hb_list** is an array where the atoms generated so far are stored.

```
SEARCH() {
while (proof not found) {
    performs IDDFS
    if (PROOF_FOUND message arrives during search) return
    if (generated clause is a unit clause) {
        if (already in hb_list and the redundancy factor is exceeded) do not add
        copy into hb_list[nexthb]
        keep track of the next atom which is shortest with minimum variables }
    while (a given time and hb_list is not empty ) {
        if (PROOF_FOUND message arrives) return
        if (SEND_ATOM message arrives) {
            pack the next atom information and send it
            break;}}}
}
```

Figure 5.8: Search algorithm collecting unit clauses at the master

5.4 PrHERBY : Parallel Semantic Tree Prover with Resolutions

We implemented a system named PrHERBY which embodies the ideas presented in earlier chapters (Figure 5.10). The slaves are based on HERBY with the atom receiving algorithm. The master is also based on HERBY but mainly carries out IDDFS and communications with slaves. It uses PVM (Parallel Virtual Machine environment) [GB94] for message passing.

The master spawns the number of slaves given by the command line argument, reads input clauses, and increases the number of base clauses, if possible, by performing the BCR (Base Clause Resolution) heuristic (Figure 5.11). Before spawning slaves

```
ASH_Parallel () {
    Send the master SEND_ATOM message asking for an atom
    while (time is not exceeded) {
        if (a message with an atom arrives) {
            unpack the atom information
            install the atom as the next atom
            break;}}
}
```

Figure 5.9: Atom receiving algorithm of a slave

and performing IDDFS, the master tries to make a closed semantic tree using a few atoms which apparently close branches. Simple theorems are solved at this stage. If no more atoms are generated, then the master carries out IDDFS during which it gathers atoms and regularly checks for messages from slaves. Atoms collected are sent to slaves according to the order of message arrival. If a slave finds a proof, it immediately sends a message to the master. The master stops the search as soon as it receives this message, collects outputs including the used atoms from the slave that found a closed semantic tree and data for performance measures, and kills all slaves in order to start a new problem. Even the master can find a contradiction during IDDFS.

After the master spawns slaves, the slaves process input clauses and increase the number of base clauses, if possible, by performing BCR (Base Clause Resolution) heuristic (Figure 5.12). Various atom selection heuristics are then tried to find an atom. Those that obviously close branches are tried first. A semantic tree is constructed by resolving the selected atom with clauses on the path to the root. If the tree contains two contradictory unit clauses, the branch is closed. The slave then backtracks up the tree and tries to close the remaining branches.

If atoms can not be generated by a slave with ASH1 to ASH4, the slave sends a

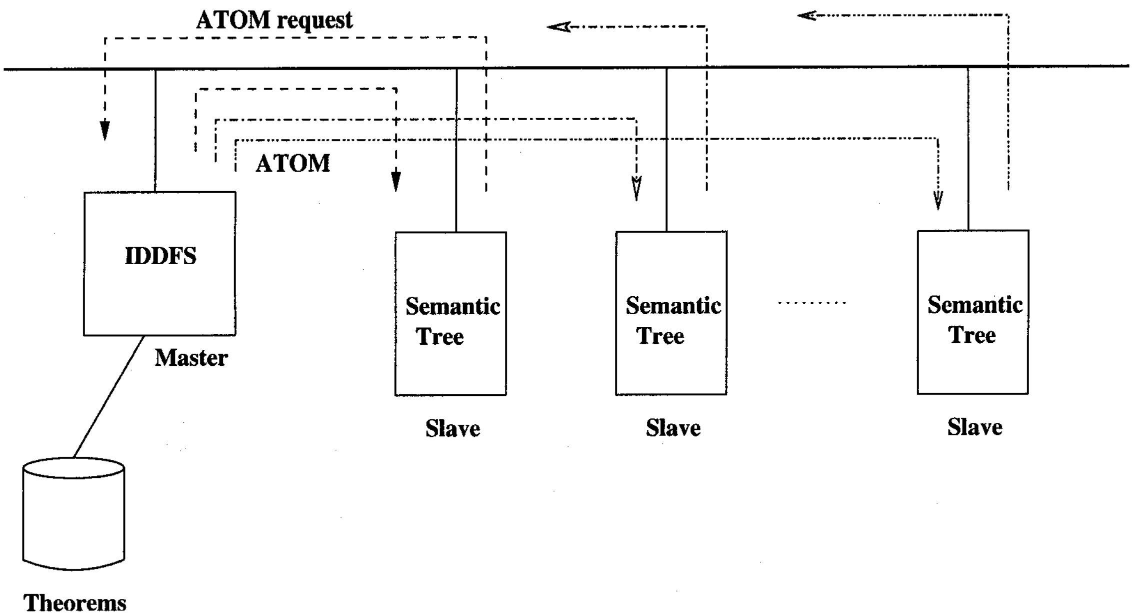


Figure 5.10: System architecture

message to the master asking for an atom. The ASH_Parallel heuristic in Figure 5.9 handles this situation. As soon as the master receives the message, it sends an atom to the slave. Each slave performs the parallel grounding scheme that instantiates the given atom. ASH5 is the last resort if no atom is chosen with the previously performed heuristics. New atoms continue to be selected until a proof is found, the predetermined time is reached, or no more atoms can be generated.

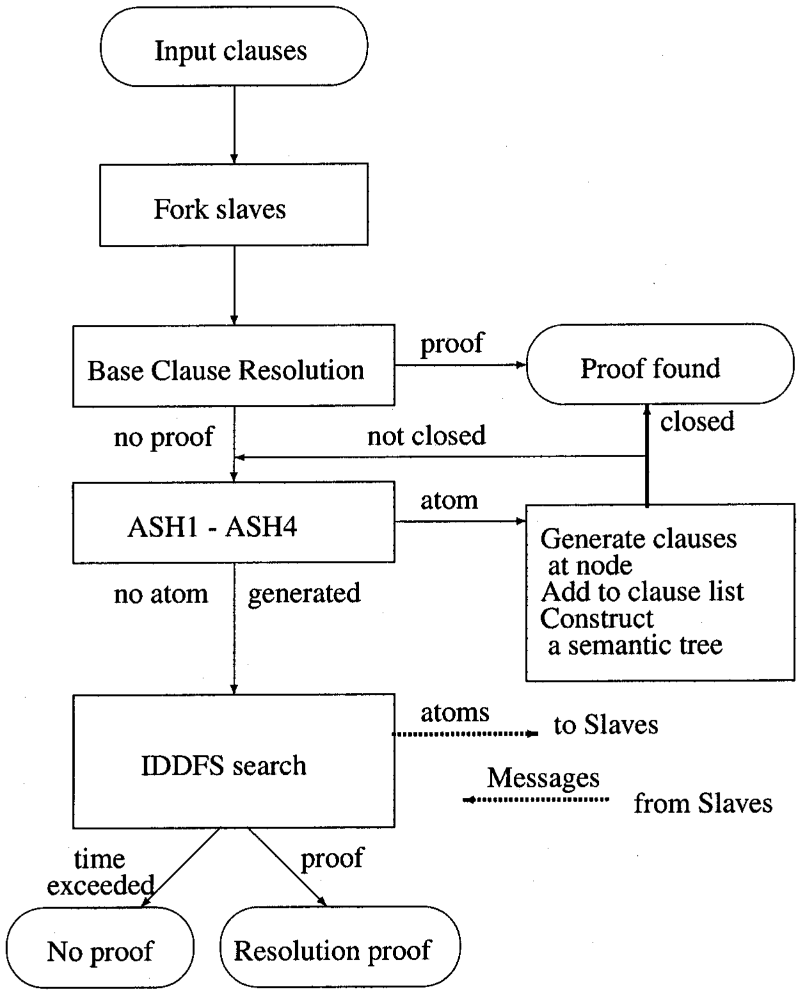


Figure 5.11: Flow of control in Master

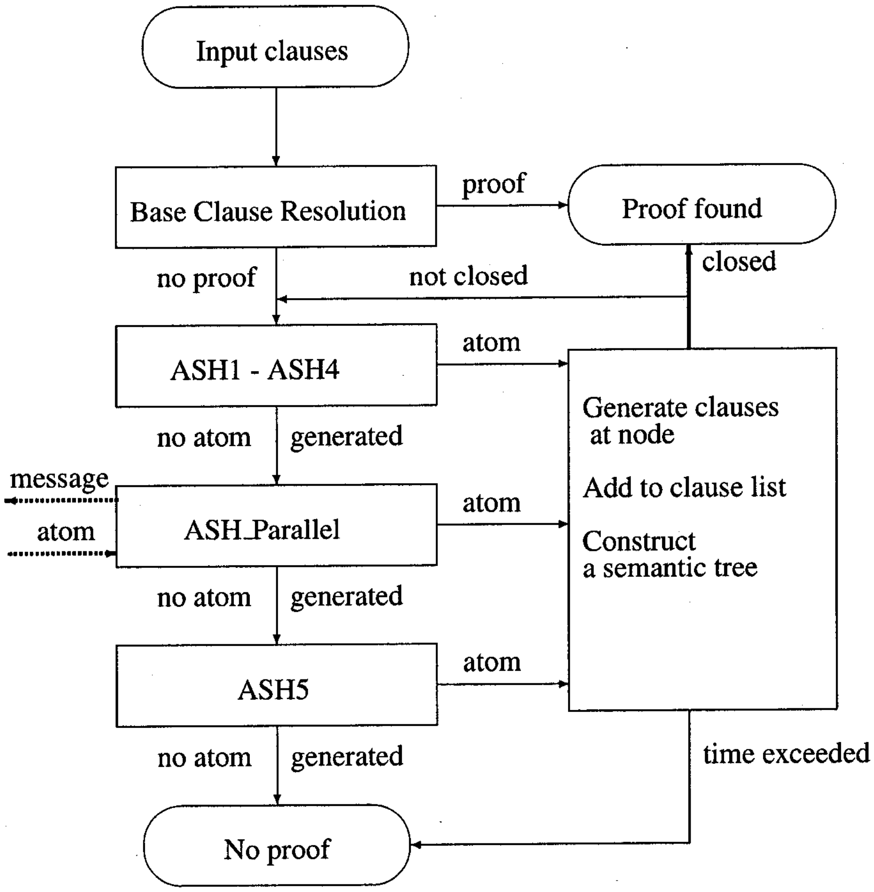


Figure 5.12: Flow of control in Slave

Chapter 6

The Experiments and Results

We now present the results of experiments using the PrHERBY system with various numbers of slaves. After introducing the test environment, we compare the results with HERBY's and then with PHERBY's.

6.1 Test environment

The Stickel set has been considered for a long time as a suitable test environment of many theorem provers having a large number of theorems with conditions such as accessibility, domain diversity, and varying difficulties.

However, HERBY is now able to solve 79 out of the 84 theorems in the Stickel test set. PrHERBY performs better than HERBY and is able to solve 83 of the 84 theorems. Among the five unsolved theorems by HERBY, PrHERBY is able to solve four: S44WOS20, S46WOS22, S52WOS28, and S19APABH. Both PrHERBY and HERBY were unable to solve theorem S50WOS26.

Since the theorems in the Stickel set are relatively easy to prove, it is hard to distinguish performance variations between systems. In this experiment, therefore, we used a subset of the TPTP library. The TPTP (Thousands of Problems for Theorem Provers) [SS98] library is a rich source of theorems developed to make the testing and evaluation of automated theorem proving systems more meaningful.

The theorems in the set come from a wide range of mathematical areas. These theorems vary widely in terms of the number of axioms each contains (from only a few to several hundred), the number of literals that each clause contains (from no more than a couple to as many as twenty), and the number of the character length of each literal (from only several to as many as forty to fifty). Due to its diversity, single strategy alone can not solve all of them. The appropriate strategy for one subclass is often very different from the one appropriate for another [New00].

From the TPTP library, we used the selected subset of 420 theorems in the CADE-14 (Conference on Automated Deduction) competition (Appendix A). Using the same time parameter as in the competition, 300 seconds, we carried out a series of experiments.

Among various LINUX based workstations with Pentium II or III CPUs ranging from 350 MHz to 1 GHz, and memories ranging from 128MB to 512MB, we chose fifteen 800MHz Pentium III machines with 256MB memories to obtain uniform test results. We varied the number of processors used by PrHERBY and presented the results of 5, 10 and 15 machines. Results from 30 machines were also presented for a reference. We used all kinds of available machines to run PrHERBY with 30 machines. In fact, the machines added additionally mostly have lower specifications than the 15 homogeneous machines. However, the results are not compared with other configurations except one for a reference.

We performed the experiments at night when the other system activities are minimal in order to maintain consistent CPU load and memory usage, and therefore minimize variables that might affect the analysis.

A successful proof by PrHERBY depends on many factors. Experimental results are obtained by considering these. A few of the most crucial ones, based on the testing experience, are listed below:

1. Frequency of atom selection using the ASH4 heuristic. Some theorems cause

PrHERBY to use ASH4 too often without success, preventing the selection of other atoms.

2. Intervals in which to use the unit-list-passing heuristic. The heuristic is CPU intensive and often spends time that could be better used in performing other computations.

3. The number of the same atoms when the master collects them. The number increases as more slaves become involved.

4. The relative involvement of each selection heuristic. Atoms selected by ASH_Parallel are used most often. However, the atoms selected by ASH5a, ASH5b and ASH5c should be used with reasonable frequency to improve success.

6.2 Comparison with HERBY

The results summarized in Table 6.1 show significantly improved performance of PrHERBY over HERBY in each category of the MIX division of the CADE-14 competition. The results of HERBY and PrHERBY with 30 machines are presented in Appendix A in detail. The competition is divided into several divisions according to problem and participating system characteristics. According to the competition, the MIX division is for mixed CNF(Conjunction Normal Form) Really-Non-Propositional Theorems [Sut97]. Mixed refers to Horn and non-Horn problems, with or without equality, but not including unit equality problems. Really-Non-Propositional means that the Herbrand universe is infinite. The MIX division is divided into four categories:

1. The HNE Category: Horn with No Equality (128[1])

2. The HEQ Category: Horn with Equality (106)

[1]Number of theorems

Table 6.1: System performance by theorem category.

Category	Thms	HERBY	PrHERBY(5)	PrHERBY(10)	PrHERBY(15)	PrHERBY(30)
HNE	128	20	49	50	50	50
HEQ	106	14	24	26	31	32
NNE	12	5	8	8	8	8
NEQ	174	58	76	77	80	92
Total	420	97	157	161	169	182

3. The NNE Category: Non-Horn with No Equality (12)

4. The NEQ Category: Non-Horn with Equality (174)

These results are displayed by the theorem category used, the number of theorems in each group, and the number of theorems solved by HERBY and PrHERBY. Note that the number in parentheses beside each PrHERBY indicates the number of machines that PrHERBY used.

As the data show, the results of PrHERBY with 5, 10 and 15 machines show improvement solving 60, 64 and 72 more theorems each than HERBY had been able to solve. Besides the overall performance, PrHERBY with 10 and 15 machines on HNE and NNE category did not show any improvement after 5 machines. The NNE category has too small size (a total of twelve theorems) to draw a conclusion. The HNE category, however, were mostly solved by the master before slaves found a proof as indicated in the output data in Table 6.5 or Appendix A. Resolutions-refutation is more appropriate than the semantic tree approach to attack the theorems in HNE group in this case. Particularly, PrHERBY with 30 machines, PrHERBY(30), outperformed HERBY by solving a total of 85 more theorems in the set of 420. The result shows an outstanding 87% (97 vs. 182) improvement over HERBY. In all cases, PrHERBY solved significantly more theorems than HERBY.

6.2.1 Speed-up of the parallel systems

ATP systems are usually evaluated in terms of

- the number of problems solved, and
- the number of problems solved with a solution output, and
- the average runtime for problem solved

in the CADE ATP system competition.

In order to evaluate PrHERBY, besides the criteria above, we compare the same amount of computing power applied to the sequential algorithm in terms of speed-up. We compare the run-times of PrHERBY with HERBY for theorems to be proven. The run-time of HERBY for a given theorem is denoted by $\mathcal{T}_s$. The run-time of PrHERBY, $\mathcal{T}_p$, is the time until one of the processors has found a proof. Based on the run-times, we define the speed-up S for a given theorem in the usual way: $S = \frac{\mathcal{T}_s}{\mathcal{T}_p}$

The CPU time of HERBY and slaves of PrHERBY is very close to the wall clock time. In the case of the master, however, the CPU time is less than the wall clock time because a portion of it is consumed as the system time to communicate with slaves. The analysis of the system time is presented in section 7.2.1. In the calculation of the speed-up, CPU times are compared instead of the wall clock time. If the master found a proof, the CPU time includes the system time for fair comparison.

In contrast to many parallel algorithms (for example, numeric computation), the speed-up obtained varies widely from theorem to theorem. This is due to the complex behavior of the search algorithm that depends on the theorems. Especially, the parallel strategy proposed here makes the speed-up vary by utilizing the atom selection of the master. The generated unit clauses can be used to close open branches of the semantic tree much earlier, thus accelerating the closure. This reduces the amount of necessary search dramatically thus increasing the speed-up values.

Among experimental data, we used 77 theorems for speed-up comparisons. Those were theorems that were solved by all configurations of machines including HERBY.

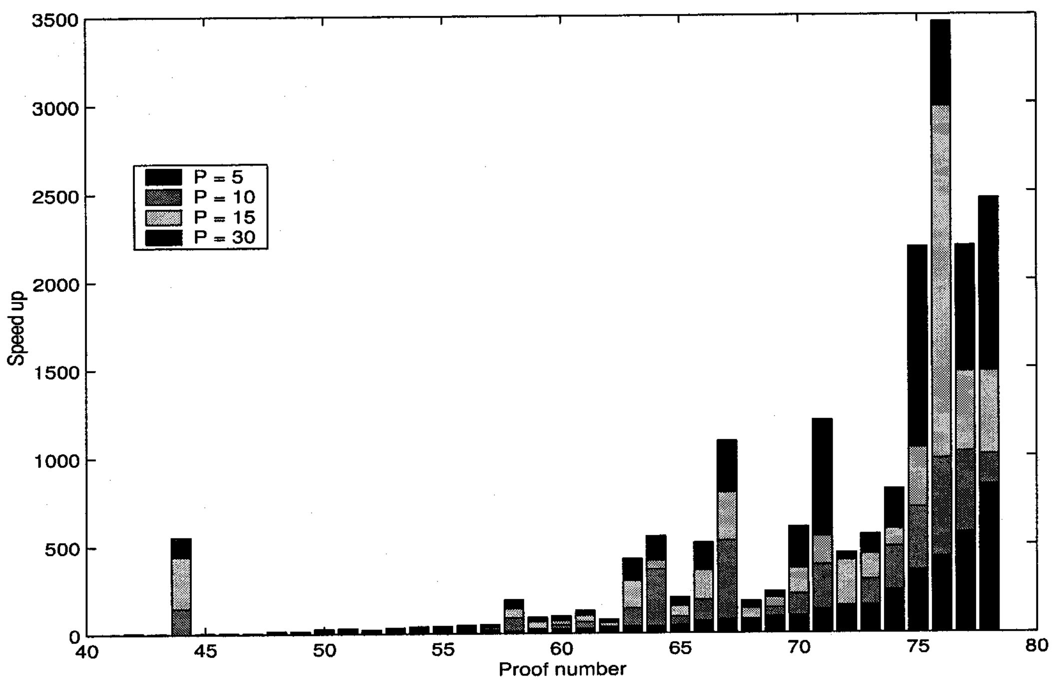


Figure 6.1: PrHERBY: Stacked speed-up values for different numbers of machines P (sorted by $P = 5$)

Although HERBY solved 97 theorems, there were theorems unsolved by some set of machines.

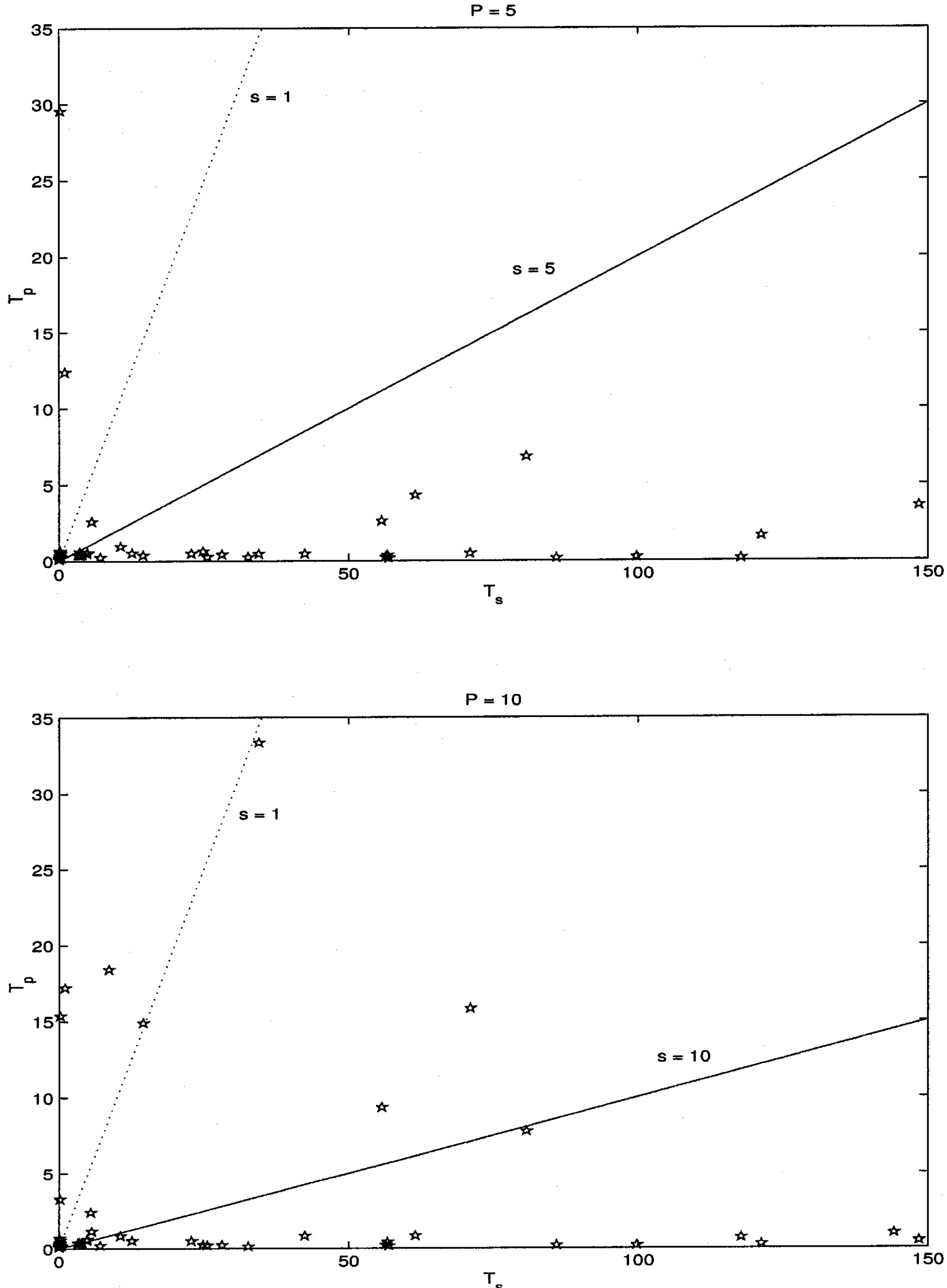
Figure 6.1 shows speed-up values of each theorem obtained from the configurations of different numbers of machines P. We produced a vertical stacked bar chart for clarity. The data are sorted by $P = 5$ speed-up values. The values of $P = 10$, $P = 15$ and $P = 30$ are added on top of them. We do not present the results of the stable front part, which shows small incremental speed-ups without any irregularities. Sometimes the speed-ups are huge and the variance is very high. There are cases that theorems solved with fewer machines are not solved with more machines or theorems are solved faster with fewer machines.

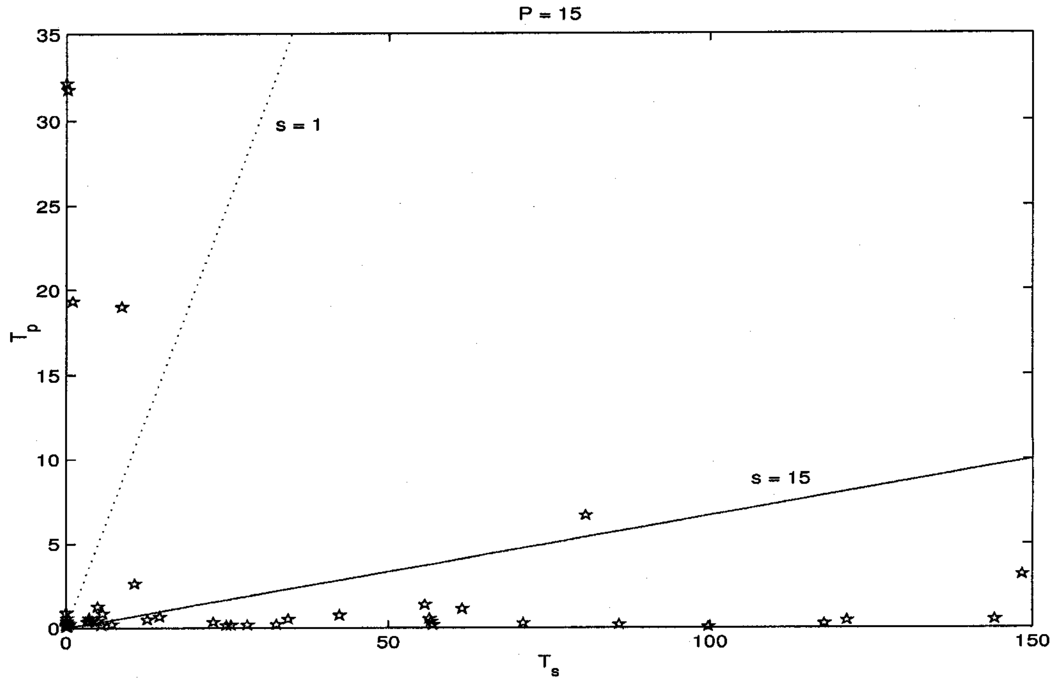
Figure 6.2 presents another view of the speed-up values. It shows the ratio of $\mathcal{T}_p$ over $\mathcal{T}_s$ for each theorem using different numbers of machines. The dotted line corresponds to $S = 1$ and the solid line to $S = P$, where P is the number of machines. The area above the dotted line contains theorems where PrHERBY is slower than HERBY. The area below the solid line contains theorems which yield a super-linear speed-up, that is, $S > P$. For the graph of $P = 5$, three points with the speed-up values less than 1 are not showed to match the scale to other graphs.

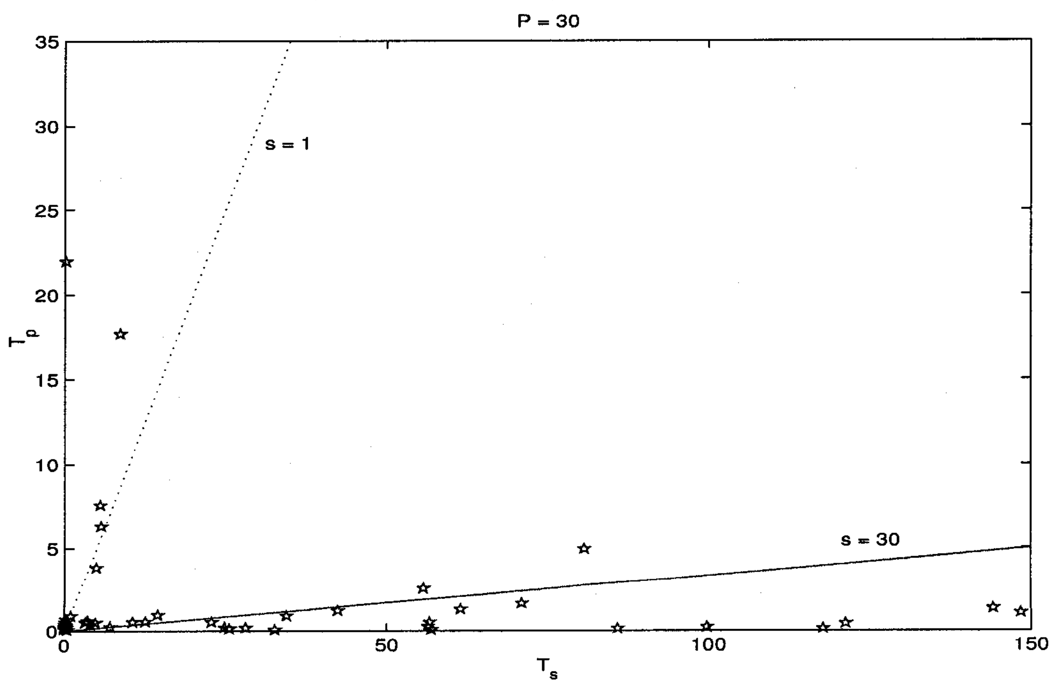
The figure presents that many theorems show super-linear speed-up. There are several cases in which PrHERBY is running slower than HERBY. In these cases, we think, the unit clauses generated were not effective to the proof of the theorem or misled the search of the proof.

The overall mean values for the speed-up are summarized in Table 6.2. In general, it is rather difficult to give a good estimation of a mean value for the speed-up over a set of examples, especially in cases where the speed-up shows a high variance. For our measurements, we considered two common mean values: arithmetic and geometric mean. The arithmetic mean is often too optimistic, resulting in a mean value too large, because a few large values of speed-up are taken into account too much. On the other hand, the geometric mean is often considered appropriate because it yields results that

Figure 6.2: PrHERBY's run-time $\mathcal{T}_p$ over HERBY's run-time $\mathcal{T}_s$ and different numbers of machines P (continued on next page)




P = 15
s = 1
s = 15
T_p
T_s
0
5
10
15
20
25
30
35
50
100
150


P = 30
s = 1
s = 30
T_p
T_s
0
5
10
15
20
25
30
35
50
100
150

are not biased to a few extreme values.

For the 77 theorems solved by all configurations, the geometric mean from $P = 15$ to $P = 30$ shows that the speed-up is decreased. In this case, direct comparison is not appropriate because not all 30 machines are the same. As another explanation, it seems that the high variance of the speed-up values that caused the increase in the arithmetic mean value was mitigated in the geometric mean.

Measuring the mean values for those theorems solved by all configurations is somewhat misleading because it does not take into account the prover's other strong points such as the number of solved theorems. To compensate for this, we calculated the speed-up values again for the theorems that were solved by at least one configuration this time. For the unsolved theorems, the maximum CPU time, 300 seconds, is applied.

For those 194 theorems except for those theorems that were solved immediately with zero second CPU time (to calculate the geometric mean), the mean values show the very high speed-up values that increases as the machines are added. In this case, the theorems solved by PrHERBY but not by HERBY(assumed 300 seconds), especially if it takes a few seconds, contributed to the large speed-up shown in Table 6.2

Table 6.2: Mean values of speed-up for different numbers of machines P

Mean	PrHERBY(5)	PrHERBY(10)	PrHERBY(15)	PrHERBY(30)
For 77 theorems solved by all configurations				
Geometric mean	2.05	2.15	2.83	2.71
For 194 theorems solved by at least one configuration				
Geometric mean	4.79	6.01	9.53	10.94

6.2.2 Results of the two strategies: parallel chained-resolution grounding and unit-list passing

Table 6.3 shows the performance improvement resulting from the parallel chained-resolution grounding strategy. Note that groups with zero entries have been removed from the table. The strategy applies whenever an atom comes from the master. Each slave has a different instance of the same atom, if possible, that the master delivers many times. The table shows how many theorems were solved before and after applying the strategy. It shows also the differences of the both results. `after` column shows the experiment that has been performed with both strategies enabled. `before` column indicates the results of PrHERBY without the grounding or unit-list passing strategy.

The parallel chained-resolution grounding strategy not only enhances overall performance but also increases system scalability. Before using it, the system scalability was weaker; as a result, theorems that had been solved using a small number of slaves failed to be solved more often using a bigger number.

Table 6.3: Effect of the parallel chained-resolution grounding (PrHERBY (15))

	BOO	CAT	COL	GEO	GRP	HEN	LCL	LDA	NUM	RNG	ROB	SET	Total
before	3	22	1	8	3	6	44	1	9	3	1	61	162
after	4	22	1	10	3	6	46	1	9	4	1	62	169
+/-	+1			+2			+2			+1		+1	+7

Table 6.4 presents the results before and after applying the unit-list passing heuristic. The performance is not as effective as the grounding strategy but still shows some enhancement to the system.

Table 6.4: Effect of the unit-list passing heuristic (PrHERBY (15))

	BOO	CAT	COL	GEO	GRP	HEN	LCL	LDA	NUM	RNG	ROB	SET	Total
before	4	24	1	8	3	5	45	1	9	4	1	61	166
after	4	22	1	10	3	6	46	1	9	4	1	62	169
+/-		-2		+2		+1	+1					+1	+3

6.3 Comparison with PHERBY

In this section, we compare the performance of PrHERBY with PHERBY. PHERBY [AN98] is the first parallel version of HERBY by Almulla. Since he participated in the CADE-15, the performance data was not revealed anymore except a brief summary of the performance of PHERBY with demodulation [Ala00]. In order to compare performance, we obtained the source code of PHERBY with demodulation and performed experiments under the same conditions with PrHERBY. According to the system description of the competition, PHERBY uses a bigger set of heuristics than HERBY, as well as cooperative computing in determining the quality of atoms to be used.

6.3.1 Comparison of solved theorems

The table 6.5 shows the theorems that PHERBY or PrHERBY solved with 15 machines within 300 seconds. We presented CPU and wall clock times. We also measured a number of atoms needed to construct a closed semantic tree. In the case of PrHERBY, "NA" in ATOMs column indicates that the master solves the theorem. Therefore, the number of atoms is not available. "-" indicates that the theorem is not solved. These data are the average of two runs. CPU time has been truncated to two significant digits but the wall clock time is measured in integer values. There are cases that the wall clock time is less than the CPU time because of the truncation.

Each system solves not only quite a different number of theorems, but also different kinds of theorems although they are based on the same HERBY. In particular, PrHERBY obtains a number of solutions in HNE category through resolutions by the

Table 6.5: Performance comparison of PHERBY and PrHERBY with 15 machines (*CPU : CPU time in sec, WC : Wall clock time in sec)*

No	Theorem	PHERBY(15)			PrHERBY(15)		
		CPU	*WC*	*ATOMs*	*CPU*	*WC*	*ATOMs*
1	GRP048-2	8.98	12	45	7.17	8	32
2	LCL006-1	9.76	15	77	56.93	66	NA
3	LCL009-1	-	-	-	17.88	20	NA
4	LCL010-1	-	-	-	0.24	0	NA
5	LCL011-1	-	-	-	5.41	6	NA
6	LCL022-1	0.16	5	31	8.97	10	NA
7	LCL023-1	-	-	-	17.57	21	NA
8	LCL025-1	59.86	61	74	1.63	2	25
9	LCL029-1	85.66	88	59	85.85	100	NA
10	LCL033-1	-	-	-	0.10	0	NA
11	LCL042-1	16.00	18	22	-	-	-
12	LCL045-1	0.14	1	12	32.80	35	40
13	LCL064-1	0.28	6	20	0.73	0	24
14	LCL075-1	-	-	-	12.94	15	NA
15	LCL083-1	-	-	-	100.52	125	NA
16	LCL086-1	-	-	-	4.27	6	NA
17	LCL087-1	-	-	-	1.26	1	NA
18	LCL088-1	-	-	-	75.63	88	NA
19	LCL101-1	-	-	-	0.96	1	NA
20	LCL102-1	-	-	-	147.26	153	NA
21	LCL104-1	-	-	-	1.64	2	NA
22	LCL107-1	-	-	-	0.46	0	NA
23	LCL108-1	-	-	-	3.50	4	NA
24	LCL110-1	-	-	-	238.64	294	NA
25	LCL111-1	-	-	-	0.33	0	NA
26	LCL118-1	-	-	-	4.94	6	NA
27	LCL120-1	-	-	-	1.28	2	NA
28	LCL130-1	-	-	-	0.06	0	NA
29	LCL182-1	0.40	10	29	7.09	8	44
30	LCL187-1	0.16	0	14	0.13	0	2
31	LCL192-1	0.15	1	15	0.12	0	4
32	LCL194-1	0.14	1	10	0.12	0	4
33	LCL195-1	0.15	1	23	0.53	0	16
34	LCL196-1	64.49	66	89	-	-	-
35	LCL198-1	-	-	-	207.29	209	125
36	LCL201-1	0.15	2	21	3.85	4	28
37	LCL204-1	0.16	2	21	4.96	5	46
38	LCL207-1	0.14	2	20	0.11	0	8
39	LCL208-1	0.15	2	25	0.43	0	19
40	LCL210-1	0.78	6	27	0.17	0	12
41	LCL211-1	0.16	2	22	0.11	0	5
42	LCL213-1	0.15	2	20	0.23	0	12

(continued on next page)

No	Theorem	PHERBY(15)			PrHERBY(15)		
		CPU	*WC*	*ATOMs*	*CPU*	*WC*	*ATOMs*
43	LCL214-1	0.16	2	25	0.15	0	11
44	LCL215-1	0.15	2	26	0.19	0	12
45	LCL216-1	0.16	2	25	0.09	0	7
46	LCL217-1	0.15	1	20	0.13	0	8
47	LCL218-1	0.16	2	24	0.12	0	9
48	LCL230-1	0.16	2	28	0.16	0	10
49	LCL231-1	0.15	2	27	0.56	0	24
50	NUM002-1	0.15	7	10	0.14	0	5
51	NUM003-1	0.14	1	8	0.13	0	7
52	NUM004-1	0.15	7	15	0.12	0	4
53	PLA007-1	112.11	113	53	-	-	-
54	PLA016-1	147.49	151	66	-	-	-
55	PLA019-1	21.18	23	36	-	-	-
56	PLA022-1	9.81	15	31	-	-	-
57	PLA022-2	8.61	14	29	-	-	-
HNE category		Solved 36/128			Solved 50/128		
58	BOO004-1	0.15	5	11	0.81	0	17
59	BOO009-1	3.08	10	36	6.49	6	36
60	BOO010-1	0.16	6	17	74.71	76	44
61	BOO012-1	1.06	6	31	3.20	3	26
62	BOO016-1	136.90	142	74	-	-	-
63	CAT001-4	0.15	1	10	0.14	0	10
64	CAT002-1	0.14	8	30	1.37	1	40
65	CAT002-4	0.16	2	48	0.15	0	9
66	CAT003-1	-	-	-	266.48	268	145
67	CAT003-2	0.15	2	2	12.41	45	53
68	CAT003-4	0.15	1	5	0.14	0	6
69	CAT004-1	0.24	7	28	32.21	33	100
70	CAT004-4	0.16	1	22	0.39	0	23
71	CAT005-4	0.25	6	22	3.21	3	50
72	CAT006-4	0.48	7	25	0.44	0	26
73	CAT009-1	7.41	8	60	0.24	0	20
74	CAT009-4	5.84	8	34	2.40	7	47
75	CAT010-1	6.81	8	66	0.19	0	16
76	CAT011-4	224.68	227	121	-	-	-
77	CAT014-4	-	-	-	34.72	35	80
78	CAT018-1	0.16	2	13	0.16	0	14
79	COL002-3	-	-	-	1.18	1	NA
80	GRP012-3	8.25	10	40	0.51	0	19
81	HEN003-3	3.47	8	40	0.10	0	13
82	HEN005-1	95.10	96	48	-	-	-
83	HEN005-3	-	-	-	28.01	28	92

(continued on next page)

No	Theorem	PHERBY(15)			PrHERBY(15)		
		CPU	*WC*	*ATOMs*	*CPU*	*WC*	*ATOMs*
84	HEN008-1	0.16	1	21	4.59	4	48
85	HEN008-3	0.78	8	33	0.21	0	20
86	HEN009-5	-	-	-	12.02	12	43
87	HEN012-3	5.38	8	77	0.28	0	21
88	LDA003-1	4.44	15	61	0.38	0	22
89	RNG006-3	16.83	19	34	113.04	114	58
90	RNG037-1	1.08	6	21	0.59	0	25
91	ROB016-1	-	-	-	0.93	1	22
HEQ category		Solved 28/106			Solved 31/106		
92	ANA002-2	136.15	138	69	-	-	-
93	SET005-1	0.15	12	10	0.11	0	7
94	SET007-1	0.16	8	12	0.14	0	9
95	SET011-1	0.15	5	7	0.12	0	10
96	SET012-1	0.15	8	21	39.35	40	44
97	SET013-1	26.68	30	60	64.00	88	NA
98	SET014-2	0.16	2	9	0.18	0	9
99	SET015-1	199.57	202	58	56.37	78	NA
100	SET055-6	0.19	6	11	0.20	0	2
NNE category		Solved 9/12			Solved 8/12		
101	CAT001-3	0.15	6	10	0.27	0	15
102	CAT002-3	0.16	4	35	0.14	0	7
103	CAT003-3	0.15	4	5	0.22	0	15
104	CAT004-3	0.15	3	13	1.20	2	32
105	CAT005-3	12.04	16	29	-	-	-
106	CAT006-3	13.99	17	37	1.73	2	35
107	CAT009-3	46.08	50	45	-	-	-
108	CAT011-3	-	-	-	7.61	8	58
109	CAT014-3	-	-	-	12.51	13	69
110	GEO002-2	-	-	-	17.11	23	NA
111	GEO006-1	26.02	28	41	92.15	92	38
112	GEO011-1	-	-	-	46.46	46	56
113	GEO026-2	-	-	-	1.46	2	25
114	GEO030-2	5.46	8	36	1.83	2	34
115	GEO036-2	-	-	-	17.30	18	49
116	GEO039-2	-	-	-	2.75	3	22
117	GEO040-2	0.17	4	17	0.17	0	10
118	GEO059-2	-	-	-	56.45	57	59
119	GEO077-4	-	-	-	133.71	135	33
120	GRP008-1	0.16	10	6	0.16	0	7
121	GRP039-4	265.16	268	55	-	-	-
122	NUM009-1	0.33	2	7	0.41	0	4
123	NUM042-1	-	-	-	0.95	1	10
124	NUM139-1	0.20	2	23	0.17	0	1

(continued on next page)

No	Theorem	PHERBY(15)			PrHERBY(15)		
		CPU	*WC*	*ATOMs*	*CPU*	*WC*	*ATOMs*
125	NUM180-1	1.61	21	24	0.75	0	7
126	NUM183-1	0.20	2	37	1.41	1	13
127	NUM228-1	0.20	6	23	0.18	0	1
128	RNG040-1	0.16	2	3	0.19	0	9
129	RNG041-1	0.22	2	15	0.15	0	4
130	SET019-4	-	-	-	0.69	0	10
131	SET024-4	-	-	-	0.47	0	6
132	SET025-4	0.21	24	6	0.29	0	4
133	SET025-9	21.03	24	16	53.18	62	NA
134	SET027-4	0.20	10	7	0.25	0	4
135	SET031-4	0.20	16	10	-	-	-
136	SET050-6	0.18	6	30	0.14	0	4
137	SET051-6	0.19	5	30	0.18	0	3
138	SET062-6	5.48	8	16	0.50	1	9
139	SET063-6	7.73	34	21	13.51	14	17
140	SET064-6	7.30	22	23	35.51	36	26
141	SET067-6	-	-	-	46.49	46	27
142	SET076-6	13.98	17	39	-	-	-
143	SET078-6	0.39	8	13	0.22	0	5
144	SET080-6	0.71	3	14	0.59	0	12
145	SET081-6	0.18	6	2	0.22	0	4
146	SET083-6	-	-	-	11.17	11	15
147	SET084-6	153.79	156	56	101.92	102	21
148	SET085-6	89.56	94	59	0.50	0	8
149	SET093-6	0.19	3	13	0.21	0	6
150	SET094-6	9.16	12	47	-	-	-
151	SET095-6	57.16	62	27	100.31	101	25
152	SET101-6	0.61	3	14	0.41	0	10
153	SET102-6	0.18	5	14	0.23	0	4
154	SET108-6	0.18	19	1	0.11	0	2
155	SET117-6	0.17	10	13	0.19	0	4
156	SET125-6	-	-	-	23.64	24	21
157	SET153-6	0.62	186	19	26.15	27	15
158	SET167-6	0.19	5	26	3.07	4	14
159	SET168-6	0.18	5	26	0.57	0	6
160	SET187-6	30.21	34	21	3.21	3	11
161	SET192-6	-	-	-	0.41	0	6
162	SET193-6	-	-	-	0.57	0	9
163	SET196-6	0.18	1	15	0.14	0	2
164	SET197-6	0.17	2	15	0.13	0	2
165	SET199-6	0.18	17	21	0.55	0	9
166	SET201-6	-	-	-	27.58	28	23

(continued on next page)

No	Theorem	PHERBY(15)			PrHERBY(15)		
		CPU	*WC*	*ATOMs*	*CPU*	*WC*	*ATOMs*
167	SET203-6	0.20	6	22	1.62	2	12
168	SET204-6	0.19	6	27	0.14	0	2
169	SET231-6	0.18	3	18	0.17	0	1
170	SET232-6	17.56	32	22	0.59	1	7
171	SET233-6	17.61	34	22	2.11	2	10
172	SET234-6	0.18	18	26	0.26	0	3
173	SET235-6	0.17	20	16	1.14	1	15
174	SET236-6	0.18	19	18	0.37	0	5
175	SET238-6	-	-	-	13.90	14	12
176	SET239-6	0.61	8	26	0.27	0	5
177	SET240-6	-	-	-	21.42	22	21
178	SET241-6	-	-	-	2.98	3	15
179	SET242-6	0.20	4	3	0.15	0	3
180	SET252-6	0.34	3	20	0.39	0	7
181	SET253-6	0.20	12	20	0.39	0	6
182	SET386-6	43.65	300	57	5.64	6	12
183	SET411-6	-	-	-	4.35	4	16
184	SET451-6	0.56	4	24	0.61	1	9
185	SET479-6	0.19	10	37	0.14	0	3
186	SET553-6	0.19	70	19	0.40	1	7
187	SET558-6	0.19	4	3	-	-	-
188	SET564-6	0.52	2	27	-	-	-
189	SET566-6	0.49	3	24	-	-	-
NEQ category		Solved 67/174			Solved 80/174		
MIX division		Solved 140/420			Solved 169/420		

Table 6.6: PHERBY's performance by theorem category

Category	Thms	PHERBY(5)	PHERBY(10)	PHERBY(15)	PHERBY(30)
HNE	128	33	36	36	36
HEQ	106	21	27	28	28
NNE	12	7	9	9	8
NEQ	174	65	67	67	68
Total	420	126	139	140	140

master. Except the total number of theorems solved, PHERBY performs as well as PrHERBY. Rather, PHERBY solved more theorems than PrHERBY if we exclude the solutions of the master. The master of PrHERBY, however, has precedence over the slaves if a proof found at the same time. From the observations, we might use instances of PHERBY as slaves for future implementation.

CPU time varies showing no particular patterns between two systems. However, WC time exaggerated with log-scaled y-axis reveals that the WC time in PHERBY takes more than PrHERBY as shown in Figure 6.3.

It shows the WC time of the theorems solved by both systems. In the case of PrHERBY, we compensated the WC time that is measured less than the CPU time due to integer truncation by adding one. PrHERBY solved the first 80 theorems immediately, but PHERBY shows significant delays to solve them. Implementation details of the systems make the difference. PHERBY controls each slave using Perl language scripts while PrHERBY implements the whole control using C. PHERBY does not utilize the whole given time solely to computations in this implementation. Its overhead to capture outputs and close a session is also larger than PrHERBY's.

Another difference of the two systems is scalability. An experiment in Table 6.6 indicates that PHERBY shows no scalability in the configurations of more than 15 machines. In fact, PHERBY configures the system by statically allocating time limit to each atom selection heuristics according to the number of slaves already available.

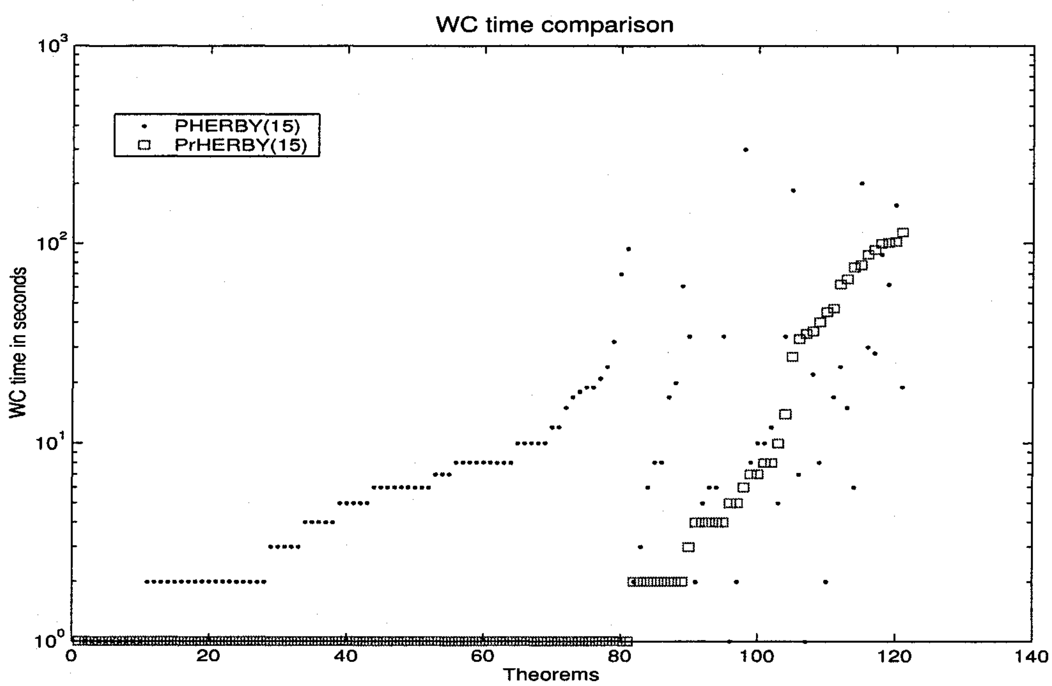


Figure 6.3: WC time comparison between PHERBY(15) and PrHERBY(15) (sorted by PrHERBY)

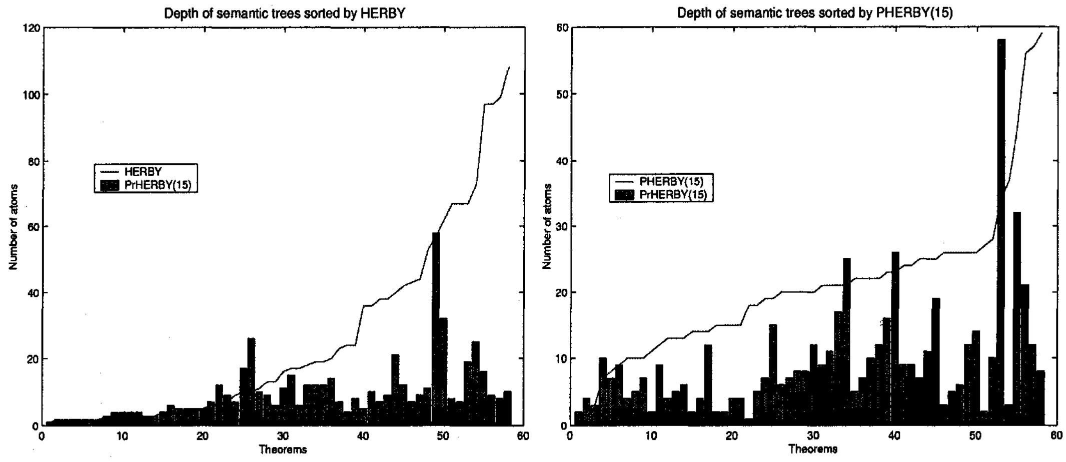


Figure 6.4: Comparison of the depth of the trees : HERBY vs. PrHERBY(15) and PHERBY(15) vs. PrHERBY(15)

6.3.2 Comparison of proof trees

As shown in the previous section 6.2.1, the speed-up values imply that PrHERBY can prove theorems effectively. As a consequence, it might generate smaller semantic trees than HERBY. To verify this, we compared the number of atoms generated for 58 theorems solved by HERBY, PHERBY(15) and PrHERBY(15) at the same time in figure 6.4. PrHERBY might find a proof using IDDFS of the master. In this experiment, the master has precedence over slaves if a proof is found at the same time. Those theorems are excluded because they do not construct a semantic tree. Two graphs compare the number of atoms of HERBY versus PrHERBY(15) and PHERBY(15) versus PrHERBY(15). Data from HERBY and PHERBY(15) were sorted according to the numbers of atoms generated, in ascending order.

We are using the atom-generation scheme in which the same atom and its negation are used at each level of a semantic tree. When more atoms are generated, the semantic tree goes deeper. The master performs a series of resolutions to select an atom to send to a slave, but these steps are performed separately in the master and do not affect

Table 6.7: Mean values of the numbers of atoms

HERBY	PHERBY(15)	PrHERBY(15)
26.0	20.9	9.6

the construction of semantic trees. If the master sends an atom to a slave, then the slave builds a semantic tree as if it had selected the atom. When 15 computers prove a theorem, one or more is going to find a better proof than the serial case and the shorter proofs are not a big surprise.

The average number of atoms in Table 6.7 clearly shows that PrHERBY generates shorter semantic trees than the other two systems. It supports our claim that the atoms sent by the master are useful enough to generally shorten the proof length.

Chapter 7

Discussion

In this chapter, we give detailed observations and analysis of the factors affecting the performance and scalability of systems. Finally, we suggest further enhancement.

7.1 System subsumption relationships

System subsumption relationship exists between systems if a system solves supersets of the problems solved by other systems [SS98b]. We can reveal the comparative strength and weakness of systems by analyzing it.

Table 7.1 lists theorems that were solved by either HERBY or PHERBY(15) but not by PrHERBY(15). "-" indicates that the theorem was not solved. As shown in the table, each group of theorems shows a very distinctive behavior. This fact exposes the diversity of the TPTP library and the experimental foundation that no single strategy most appropriately used in one subclass can solve other classes (which require very different strategies).

As examples of this diversity, our experiments show that the system subsumption relationships do not exist completely. Even though PrHERBY(15) is based on HERBY and solved far more theorems (97 vs. 169) than HERBY, nine theorems were solved by HERBY but were not by PrHERBY(15).

Table 7.1: System subsumption relationships (15 machines)

Theorem	HERBY	PHERBY	PrHERBY	Theorem	HERBY	PHERBY	PrHERBY
	Proof	Proof	Proof		Proof	Proof	Proof
LCL042-1	-	yes	-	CAT009-3	-	yes	-
LCL196-1	-	yes	-	GRP039-4	-	yes	-
PLA007-1	-	yes	-	SET031-4	-	yes	-
PLA016-1	-	yes	-	SET076-6	-	yes	-
PLA019-1	-	yes	-	SET082-6	yes	-	-
PLA022-1	yes	yes	-	SET094-6	yes	yes	-
PLA022-2	yes	yes	-	SET194-6	yes	-	-
BOO016-1	-	yes	-	SET195-6	yes	-	-
CAT011-4	yes	yes	-	SET558-6	-	yes	-
HEN005-1	-	yes	-	SET564-6	-	yes	-
ANA002-2	-	yes	-	SET566-6	yes	yes	-
CAT005-3	yes	yes	-				

Similarly, regarding the system subsumption relationship between PHERBY and PrHERBY, PrHERBY(15) did not solve twenty theorems that PHERBY(15) solved though PrHERBY did solve more theorems than PHERBY. PrHERBY otherwise subsumes PHERBY.

Along with these phenomena, the system subsumption relationships do not exist completely among PrHERBYs with different numbers of machines. In this case, the strategy of distributing a number of atoms appears to be incompetent. Atoms from the master might divert the proofs for some theorems.

IDDFS sequence prompts this behavior, because the distribution order of atoms is not inclusive. For example, let us consider the atoms collected in table 7.2. Table 7.3 indicates the atoms distributed to each slave when PrHERBY uses 1, 2 and 5 slaves.

Table 7.2: Sequence of the collected atoms at the master

$atom_1$	$atom_2$	$atom_3$	$atom_4$	$atom_5$	$atom_6$	$atom_7$	$\cdots$

Table 7.3: Sequences of the atoms received at each slave

	1 Slave	2 slaves		5 slaves				
	S_1	S_1	S_2	S_1	S_2	S_3	S_4	S_5
1^{st}	$atom_1$	$atom_1$	$atom_2$	$atom_1$	$atom_2$	$atom_3$	$atom_4$	$atom_5$
2^{nd}	$atom_2$	$atom_3$	$atom_4$	$atom_6$	$atom_7$	$\cdots$	$\cdots$	$\cdots$
3^{rd}	$atom_3$	$atom_5$	$atom_6$	$\cdots$	$\cdots$			
4^{th}	$atom_4$	$atom_7$	$\cdots$					
5^{th}	$atom_5$	$\cdots$						
6^{th}	$\cdots$							

Table 7.3 shows that the same slave takes different sets of atoms in different configurations. This different sequence of atoms does not guarantee that PrHERBY will

always find the proofs solved with fewer slaves although it will diversify attempts to find proofs. Increasing the number of the same atoms is a way to relieve the symptoms, but it should not deteriorate the overall performance with limited time constraints.

7.2 Scalability

In this section, we analyze the scalability of PrHERBY. We measured the system times for implementing the message passing mechanism between machines and the ratio of used atoms to generated atoms of the master.

7.2.1 System times

In PrHERBY, each slave utilizes most of the given time to construct semantic trees and spends negligible amount of the time on the operation of receiving atoms from the master according to experiments. On the other hand, the master spends substantial amount of the given time to distribute atoms to each slave.

PrHERBY is highly CPU-intensive. We can estimate the time spent on the message passing by measuring system times. In UNIX, the run time of a program consists of CPU and SYSTEM time and system calls to measure them are provided.

To identify the overhead of the master, we compare the system times of the 218 theorems that were not solved under any configurations. We choose the unsolved theorems because the system time of the master for solved theorems varies too much to determine any patterns the system has and the theorems are generally solved in a few seconds in many cases.

Table 7.4 shows the average system time of the master with a different number of machines. The table is divided according to theorem categories. With the increasing number of machines, PrHERBY shows increasing system times but the increment is generally very small compared to the addition of the number of machines. Even if the number of machines is doubled, the system time is increased just by 2 to 3 seconds.

Table 7.4: Mean values of the system times in seconds

category	Thms	PrHERBY(5)	PrHERBY(10)	PrHERBY(15)	PrHERBY(30)
HNE	70	29.1	31.4	32.3	34.7
HEQ	68	37.9	39.4	39.7	41.6
NNE	4	69.3	69.8	71.2	70.9
NEQ	76	28.4	30.2	30.3	31.1

Figure 7.1 shows the system time for each category. Unlike the mean values, the graphs reveal particular patterns. Among the four of them, HEQ and NEQ show smooth curves that do not present big differences in system times as the number of machines increases. NNE category has too small samples to analyze, but the data shows that the difference between 15 and 30 machines is very small.

For HNE category, the first half of the graph fluctuates according to the number of used machines. In fact, theorems of PLA (Planning) domain in the field of Computer Science show the behavior. The system time increases as the machines are added. The theorems in the domain are also hard to solve in PrHERBY experiments.

For HEQ category, theorems of HEN (Henkin Models) domain in the field of Logic spends a third of the given time on the system time.

Because the analysis is based on the unsolved theorems, other than showing the system behavior, its relevance with theorems to be solved is unknown. However, if we assume that atoms contribute more in finding a proof as more atoms are generated, we can expect that the scalability of PrHERBY can be achieved in the theorems of the group not mentioned above.

7.2.2 Used versus generated atoms

The master generates a number of atoms through resolutions. These atoms are collected and distributed upon requests from slaves. We measured what percentage of the generated atoms were used for distribution and thus consumed. For the same 218

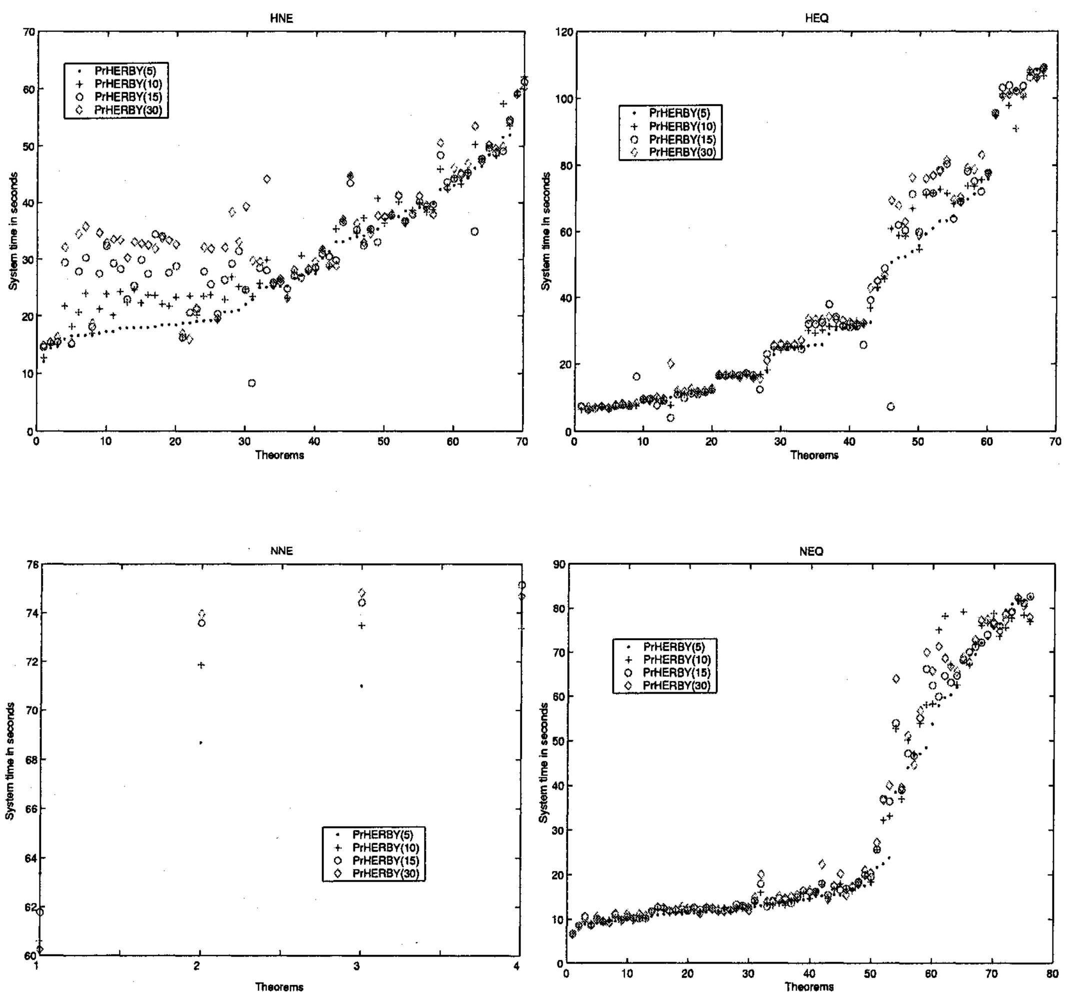


Figure 7.1: Comparison of system times of the master classified by the theorem category

unsolved theorems, Figure 7.2 compares the ratio of used atoms to generated atoms of PrHERBY obtained as follows.

$$\frac{Used\ atoms}{Generated\ atoms} \times 100$$

As the number of machines increases, the ratio gets bigger meaning that PrHERBY with more machines uses more atoms. The ratio, however, is below 10% in most cases. If the atom supply is not sufficient, many slaves are going to work on their own without taking the advantage of the competition caused by the atom distribution. In that case, the situation is the same as running several copies of HERBY. On the other hand, the system can use as many machines as possible while the atom supply lasts if the overhead is negligible as the system times in this case. As a strategy of the master, we can control the consumption rate of the generated atoms according to the number of available machines.

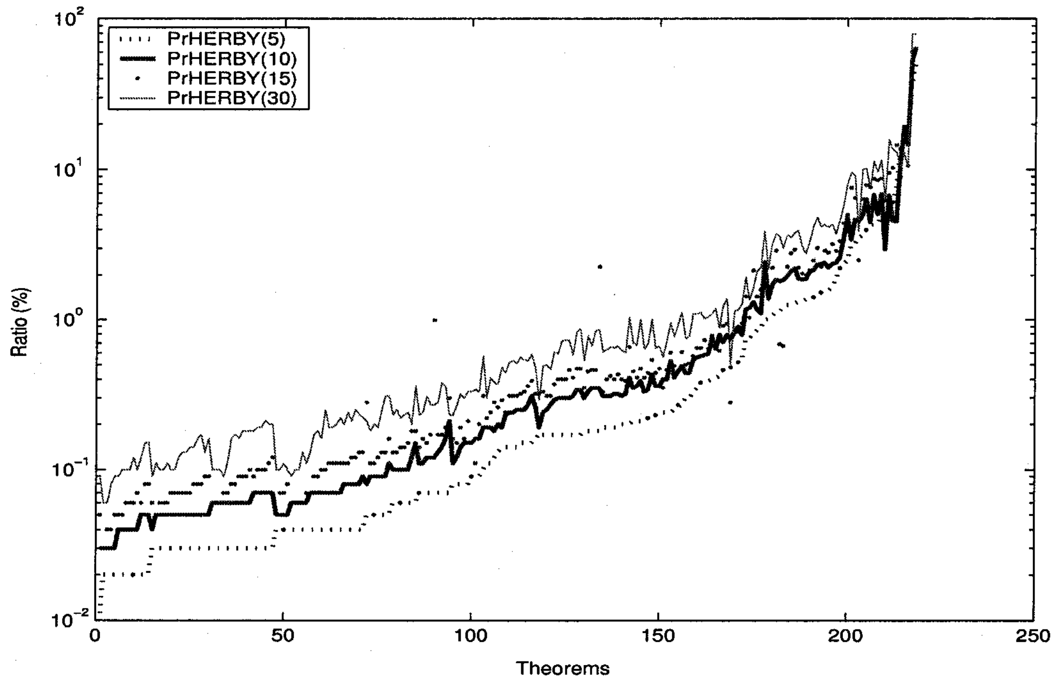


Figure 7.2: Sorted ratio of used vs. generated atoms with log-scaled y axis

The results summarized in Table 6.1 shows that performance generally improves for HEQ and NEQ categories as the number of slaves increases. From the illustration of Figure 7.1 and 7.2, we can infer the scalability of PrHERBY because the system overhead is not proportional to the number of machines and the available number of atoms is large enough.

There are thousands of atoms that can be distributed, leading to different proof attempts while the number of different strategies is far less. Unlike SiCoTHEO [Sch97], OCTOPUS [New98] and PHERBY [AN98], which are limited by the number of available strategies, PrHERBY is scalable for theorems in many domains. The more slaves become involved, the more closed semantic trees can be built for those theorems.

7.3 The number of clauses generated

In this section, we compare the number of clause generated. we choose the unsolved theorems in PrHERBY(15) to find out the system's particular behavior. The following table shows a typical output of the unsolved theorem GEO002-1. After the theorem name, the collected information of the master comes next. The information of each slave follows. We explain each parameter below.

```
 19 ../P97/GEO002-1+short+ran.p
XXXXXXXX CPU: 221.59 SYS:75.99 WC: 299 RES: 9492429  Nodes: 7036929  ATOMs: 7702 NATOM: 479
BASE : 56   REVISED CLS: 83   -GROUND: 0 -UATOM: 0
>> [S 3] CP: 296.75 WC: 298   RES: 22706 ARE: 1114720 Nod: 415 SA:158 U: 12349 ASHs:  17   1   0 119   0   6   4   4   7
>> [S 1] CP: 100.77 WC: 299   RES:  6217 ARE:  603329 Nod: 171 SA: 65 U: 16602 ASHs:   2   4   0  20   0  16   2   8  13
>> [S12] CP: 297.38 WC: 298   RES:  2299 ARE: 1684286 Nod:  74 SA: 72 U: 15703 ASHs:   0   0   0   0   0  30  12   6  24
>> [S 0] CP: 298.85 WC: 300   RES: 10327 ARE:  938520 Nod: 266 SA: 74 U: 26044 ASHs:   2   0   0  30   0  12   7   9  14
>> [S 7] CP: 298.70 WC: 300   RES:  2232 ARE: 1657773 Nod:  78 SA: 73 U: 18078 ASHs:   1   0   0   0   0  28  10  10  24
>> [S 6] CP: 298.85 WC: 300   RES:  3029 ARE: 1469343 Nod:  92 SA: 81 U: 19383 ASHs:   1   2   0   0   0  30  11  10  27
>> [S 2] CP: 298.60 WC: 300   RES:  5976 ARE: 1366756 Nod: 158 SA: 84 U: 23587 ASHs:   3   2   0  24   0  22   5   9  19
>> [S 4] CP: 298.01 WC: 300   RES:  4231 ARE:  652399 Nod: 134 SA: 64 U: 30892 ASHs:   4   0   0  27   0  15   5   2  11
>> [S 8] CP: 150.27 WC: 300   RES:  1956 ARE: 1147073 Nod:  68 SA: 60 U: 15845 ASHs:   2   0   0   0   0  21  10   7  20
>> [S10] CP: 298.82 WC: 300   RES:  1730 ARE: 1171971 Nod:  61 SA: 52 U: 21203 ASHs:   1   1   0   0   0  18   8   7  17
>> [S 9] CP: 298.19 WC: 300   RES:  4802 ARE: 1931783 Nod: 116 SA: 88 U: 16519 ASHs:   3   0   6   0   0  25  11  16  27
>> [S 5] CP: 295.53 WC: 298   RES:  4416 ARE: 2530158 Nod: 116 SA:110 U: 11105 ASHs:   0   0   0   0   0  28  24  22  36
>> [S13] CP: 298.74 WC: 300   RES:  3537 ARE: 1888162 Nod:  97 SA: 81 U: 17328 ASHs:   0   2   3   0   0  31   8  11  26
>> [S11] CP: 301.49 WC: 303   RES: 23154 ARE: 1128828 Nod: 420 SA:158 U: 11871 ASHs:  17   1   0 119   0   5   4   5   7
```

We averaged the number of clauses generated (RES), which is used for constructing semantic trees and the clause generated during atom selections (ARE). In addition, the

`RES`	the number of resolutions for IDDFS in the master and for semantic tree construction in slaves.
`ARE`	the number of resolutions generated in the atom selection procedure of each slave.
`CP, CPU`	CPU time.
`SYS`	system time.
`WC`	wall clock time.
`Nodes, Nod`	number of nodes.
`ATOMs`	generated atoms in the master.
`NATOM`	distributed atoms.
`SA`	number of atoms used for constructing a semantic tree.
`U`	unit clauses generated.
`ASHs`	frequency of the used atom selection heuristics.

number of clauses generated in the master (`RES`) are compared in Figure 7.3. The data are displayed in ascending order according to the number of clauses generated by slaves (`ARE`).

The graph shows that the master and the slaves of PrHERBY are dealing with significantly different number of clauses. It shows that the number of clauses generated for a semantic tree construction, which is denoted by `Slave`, is very small. In slaves of PrHERBY, more clauses are generated in the process of selecting atoms than constructing a semantic tree itself. On the other hand, the master generates huge number of clauses in general than the above two factors.

The graph clearly presents a uniform limit of the number of clauses that PrHERBY can reach and implies that PrHERBY spends CPU time mostly on executing atom selection procedures which involve generating clauses.

From the observation, we consider that the further aspect of performance enhancement is the efficiency of strategies. It is well known that the system equipped with the equality-handling strategy is superior in solving the theorems with equality. Merely adding it does not guarantee overall performance improvement, however, because it tends to generate superfluous clauses.

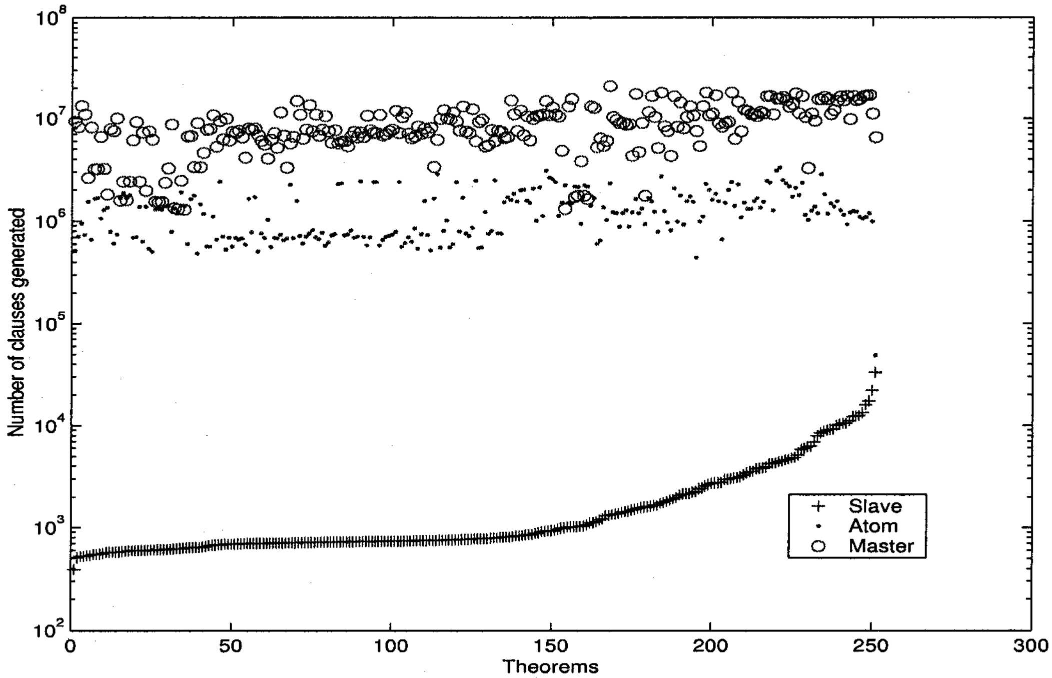


Figure 7.3: Comparison of the number of clauses generated (`Slave` : clauses generated to construct a semantic tree, `Atom` : clauses generated to search for atoms, `Master` : clauses generated by IDDFS in the master)

In PrHERBY, the atom selection heuristics have a substantial margin of improvements. Those heuristics generate many clauses for atom selections and consequently, limited amount of time is available for expanding a semantic tree deeper.

Newborn's successful prover, THEO [New01], provides an insight of the enhancement. THEO uses a large hash table to store information about clauses generated during the search. *Unit hash table resolution* in THEO deletes a literal in a clause if the negative hash code of the literal is found in the hash table. If every clause or resolvent of PrHERBY were hashed at generation time as THEO does, PrHERBY could achieve many operations in $\mathcal{O}(1)$. A clause can be shortened whenever one of its literals has a resolvable pair in the hash table, thereby increasing the likelihood of success.

7.4 Semantic tree generation vs. Resolution-refutation

Semantic tree generation is a systematic way of implementing the Herbrand's theorem. It has been argued in Almulla's thesis [Alm95] that the semantic tree generation method can grow to become no less than the other practical methods for detecting unsatisfiability. As a proof of the argument, Almulla presented several theorems for which the semantic tree generation gave far better results than resolution-refutation.

We verified whether the argument regarding the usefulness of semantic tree generation is still valid. For all examples—Pigeonhole theorem [Pel86], Arbitrary graph theorems [Pel86, Urq87], Foothold theorems, and Shoe-Boxes theorems— HERBY and THEO showed immediate proofs with differences that hardly provide any meaningful interpretation. Therefore, the assertion that there are theorems for which the semantic tree generation gives better results is no longer valid with these examples.

HERBY and THEO that Almulla had used for the tests have been improved substantially for the past several years. Especially, HERBY adapted many successful strategies of THEO. For example, atom selection heuristic 4, maintaining unit list, is an adaptation of THEO's unit resolution strategy.

Basically, there exists a correspondence between the two methods as mentioned in section 4.1. Although the performance shows that the semantic tree generation is still weaker than the resolution theorem prover, THEO, with the absolute metric of the number of theorems solved, the difference is largely related to the sophistication of the implementation, not the power of the approach.

We strongly believe that the semantic tree generation can be as good as the other methods for proving unsatisfiability, including the resolution-refutation method. Furthermore, it has exceptional possibilities of linearity and scalability as exploited in this thesis.

Chapter 8

Conclusions

In this chapter, we summarize our work and suggest possible extensions.

8.1 Summary

We have the idea of combining resolutions to the semantic tree construction because of the observation that most semantic trees tend to be thin. Linearity is favored in various theorem provers due to its simplicity and easy applicability of several strategies. Because the strategy of building linear semantic trees is incomplete, we introduced resolutions to provide closure of semantic trees and envisioned a strategy of integrating semantic trees with resolutions. This combination strategy has some prospects:

1. As the linear property of semantic trees suggests, applying resolvents to obtain closure is a promising way to lead to a proof. Applying a resolvent is legitimate, because it is a logical consequence of the resolved clauses.

2. Different strategies tend to complement each other since no single strategy performs optimally on all theorems or in every area.

3. Semantic trees and resolution-refutation proof trees are convertible to each other. Once a resolution-refutation tree is built, atoms to close the corresponding se-

mantic tree can easily be obtained.

4. Sophisticated strategies used in resolution-refutation provers can be applied to refine atom selections.

With these prospects, we effectively integrated the semantic tree construction with resolution-refutation by building a parallel semantic tree prover, PrHERBY. Among the several design alternatives of parallelization, we used iterative deepening depth-first search to explore the search space at the master side. Slaves constructed semantic trees on their own but got assistance by asking the master for atoms when their useful strategies ran out. In addition, we proposed parallel chained-resolution grounding scheme, in which each slave tries to take different instances of the same atoms.

The implemented system followed similar schemes such as strategy parallelism [WL99] and scheduling method [SW00]. The schemes were to perform several strategies competitively or try to find adequate strategies in advance. PrHERBY performed resolutions in the master in an effort to find suitable atoms. The atoms were transferred to slaves competitively in very diversified orders and with opportunities for taking different instances.

We performed experiments with the 420 CADE-14 selection list, which is a part of the TPTP library. We compared the overall performance of three systems—HERBY, PHERBY, and PrHERBY. Our experiments demonstrated that PrHERBY significantly outperformed the semantic tree theorem prover, HERBY and PHERBY, the first parallel version of HERBY. Moreover, it was usually able to build shorter closed semantic trees.

We compared the system time and the consumed number of atoms among various system configurations of PrHERBY. The comparison showed that the atoms generated in the master were consumed rapidly as the number of slaves increased but the master was still generating enough atoms. Furthermore, the system time increased very slowly as the number of slaves was increased, showing that PrHERBY is scalable in many domains.

Finally, we presented the number of clauses generated from the master, from slaves to find atoms and from slaves to construct semantic trees.

In our comparisons and discussions, we provided aspects that could be enhanced. A section is devoted to the review of the methods we discussed: semantic tree generation versus resolution-refutation.

8.2 Future work

- Further work to show a minimum proof is required. Not all the atoms selected contribute to a proof. Keeping atoms that contribute to the proof enhances understanding of it. One possibility would be to use the same IDDFS scheme to construct a semantic tree at slave side. The semantic tree would be built using the IDDFS scheme, reordering atoms selected so far according to their importance.

- Demodulation and paramodulation strategies have not been incorporated into the system. These methods are known to generate unnecessary clauses. But this disadvantage could be overcome in PrHERBY by applying the rules to atoms generated by the master. Specifically, *parallel-chained demodulation* would be advantageous, because it would give a different copy of demodulated resolvents to each slave.

- Although we experimentally measured the efficiency of the theorem provers, PrHERBY, PHERBY and HERBY, an analytical study of the efficiency of theorem proving strategies has emerged [PZ97]. Further research into the theoretical analysis of theorem proving strategies could supplement the work described here.

- The current scheme delivers atoms upon requests by slaves. Information about how atoms are generated can also be included at the time of delivery. The slave then repeats the resolutions and increases the number of clauses. This strategy is an extension of the BCR scheme but more context sensitive.

- The current parallel system is static in the sense that communication between the master and slaves is unidirectional and does not include any active information to affect the decision of atom selections. If atom selection procedures could be improved substantially, as discussed, decisive information obtained from various techniques of decision making could be used to enhance performance.

Bibliography

[Ala00] Ali Al-Anjawi, *Using demodulation in semantic tree theorem provers*, ICCI2000, Nov. 2000.

[Alm95] Mohammed Almulla, *Analysis of the use of semantic trees in automated theorem proving*, Ph.D. Thesis, School of Computer Science, McGill University, Montreal, Canada, 1995.

[AN96] Mohammed Almulla and M. Newborn, *The practicality of generating semantic trees for proofs of unsatisfiability*, International Journal of Computer and Mathematics, Vol. 62, pp. 45-61, 1996.

[AN98] M. Almulla and M. Newborn, *http://www.cs.miami.edu/~tptp/CASC/15*, 1998.

[And68] Peter B. Andrews, *Resolution With Merging*, Journal of the ACM, Vol. 15, pp. 367-381, 1968.

[Ast94] Owen Astrachan, *METEOR: Exploring model elimination theorem proving*, Journal of Automated Reasoning, Vol. 13, pp. 283-296, 1994.

[BCLM92] S. Bose, E. Clark, D. E. Long, and S. Michaylov, *PARTHENON: A parallel theorem prover for non-Horn clauses*, Journal of Automated Reasoning, Vol. 8, pp. 153-182, 1992.

[Ben92] Dan Benanav, *Recognizing Unnecessary Clauses in Resolution Based Systems*, Journal of Automated Reasoning, Vol. 9, pp. 43-76, 1992.

[BH94] Maria Paola Bonacina and Jieh Hsiang, *Parallelization of deduction strategies: an analytical study*, Journal of Automated Reasoning, Vol. 13, pp. 1-33, 1994.

[Bon20] Maria Paola Bonacina, *A taxanomy of parallel strategies for deduction*, Annals of mathematics and artificial intelligence, Vol.29, pp. 223-257, 2000.

[Cha70] C.L. Chang, *The unit proof and the input proof in theorem proving*, Journal of the ACM, Vol. 17, No. 4, pp. 698-707, 1970.

[Cha72] C.L. Chang, *Theorem proving with variable-constrained resolution*, Information Science 4, pp. 217-231, 1972.

[Chu94] Heng Chu, *Semantically guided first-order theorem proving using hyper-linking*, Ph.D. Thesis, University of North Carolina at Chapel Hill, 1994.

[CL73] C.L. Chang and R.C.T. Lee, *Symbolic logic and mechanical theorem proving*, Academic Press, 1973.

[CP93] Heng Chu and David A. Plaisted, *Model finding strategies in semantically guided instance-based theorem proving*, Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93), pp. 19-28, June 1993.

[CP94] Heng Chu and David A. Plaisted, *Semantically guided first-order theorem proving using hyper-linking*, Proceedings of CADE-12, pp. 193-206, 1994

[Ert92] W. Ertel, *OR-parallel theorem proving with random competition*, Proceedings of Logic for Programming Artificial Intelligence and Reasoning (LPAR'92), LNAI 624, pp. 226-237, 1992.

[FF97] Marc Fuchs and Matthias Fuchs, *Applying case-based reasoning to automated deduction*, Case-Based Reasoning Research and Development, second International Conference (ICCBR'97), LNCS, Vol. 1266, pp. 23-32, 1997.

[FLSY74] S. Fleisig, D. Loveland, A. K. Smiley III and D. L. Yarmush, *An implementation of the model elimination proof procedure*, Journal of the ACM, Vol. 21, pp. 124-139, Jan. 1974.

[FW98] Marc Fuchs and Andreas Wolf, *System description: Cooperation in model elimination: CPTHEO*, Automated deduction, CADE-15, LNAI, Vol. 1421, pp. 42-46, 1998.

[GB94] A. Giest, A. Beguelin et al., *PVM: Parallel Virtual Machine, A user's guide and tutorials for networked parallel computing*, MIT Press, 1994.

[GN86] Michael R. Genesereth and N. J. Nilson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1986.

[GWY99] X.S. Gao, D. Wang and L. Yang, *An application of automatic theorem proving in computer vision*, Automated Deduction in Geometry (ADG'98), LNAI, 1669, pp. 207-232, 1999.

[HS97] Joseph D. Horton and Bruce Spencer, *Clause trees: a tool for understanding and implementing resolution in automated reasoning*, Artificial Intelligence, Vol. 92, pp. 25-89, 1997.

[Hur99] Joe Hurd, *Integrating Gandalf and HOL*, The 12th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'99), LNCS, 1690, pp. 311-321, 1999.

[JO92] Anita Jindal and Ross Overbeek, *Exploitation of parallel processing for implementing high-performance deduction systems*, Journal of Automated Reasoning, Vol. 8, pp. 23-38, 1992.

[KH69] R. Kowalski and P. Hayes, *Semantic trees in automatic theorem proving*, Machine Intelligence, Vol. 4, pp. 87-101.

[Kim04] Choon Kyu Kim, *Exploiting Parallelism: Hihgly competitive semantic tree theorem prover*, International Journal of Computer Mathematics, Vol. 81, 2004, To appear.

[KN03] Choon Kyu Kim and Monty Newborn, *Competitive semantic tree theorem prover with resolutions*, Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI'03), LNCS, 2840, pp. 227-231, 2003.

[Kor85] Richard E. Korf, *Depth-first iterative-deepening: an optimal admissible tree search*, Artificial Intelligence, pp. 97-109, 1985.

[Lap98] Patrice Lapierre, *Willow: extending Herby's semantic tree theorem proving heuristics*, Master Thesis, School of Computer Science, McGill University, Montreal, Canada, 1998.

[Let92] R. Letz et al., *SETHEO: A High-performance theorem prover*, Journal of Automated Reasoning Vol. 8, pp. 183-212, 1992.

[LO85] E.L. Lusk and R.A. Overbeek, *Reasoning about equality*, Journal of Automated Reasoning, Vol. 1, pp. 209-228, 1985.

[Lov69] D.W. Loveland, *Theorem provers combining model elimination and resolution*, Machine Intelligence, Vol. 4, pp. 73-86, 1969.

[Lov69b] D.W. Loveland, *A simplified format for the model elimination procedure*, Journal of the ACM, Vol. 16, pp. 349-363, 1969.

[Lov70] D.W. Loveland, *A linear format for resolution*, Symposium on Automatic Demonstration, Lecture Notes in Mathematics, 125, Springer-Verlag, Berlin and New York, pp. 147-162, 1970.

[Lov78] D.W. Loveland, *Automated theorem proving: A logical basis*, North-Holland, 1978.

[LP92] S.J. Lee and D.A. Plaisted, *Eliminating duplication with the hyper-linking strategy*, Journal of Automated Reasoning, Vol. 9, pp. 25-42, 1992.

[LP94] S.J. Lee and D.A. Plaisted, *Problem solving by searching for models with a theorem prover*, Artificial Intelligence, Vol. 69, pp. 205-233, 1994.

[Luc67] D. Luckham, *The resolution principle in theorem-proving*, Machine Intelligence, Vol. 1, pp. 47-61, 1967.

[Luc70] D. Luckham, *Refinement theorems in resolution theory*, Symposium on Automatic Demonstration, Lecture Notes in Mathematics, 125, Springer-Verlag, Berlin and New York, pp. 163-191, 1970.

[McC97] W. McCune, *33 Basic test problems: A practical evaluation of some paramodulation strategies*, Automated reasoning and its applications, MIT Press, 1997.

[New98] M. Newborn, *http://www.cs.miami.edu/~tptp/CASC/15/SystemDescriptions .html#TGTP*, 1998.

[New00] M. Newborn, *Some thoughts related to parallel resolution-refutation theorem proving*, Sixth International Symposium on Artificial Intelligence and Mathematics, http://rutcor.rutgers.edu/~amai/aimath00, Florida, Jan. 2000.

[New01] M. Newborn, *Automated Theorem Proving: Theory and Practice*, Springer-Verlag, 2001.

[Nil71] N. J. Nilson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

[Pel86] F.J. Pelletier, *Seventy-five problems for testing automatic theorem provers*, Journal of Automated Reasoning, Vol. 2, pp. 191-216, 1986.

[Pla76] David A. Plaisted, *Theorem proving and semantic trees*, Ph.D. Thesis, Stanford University, 1976.

[PZ97] David A. Plaisted and Yunshan Zhum, *The efficiency of theorem proving strategies*, Friedrich Vieweg & Sohn, Second edition, 2000.

[Rob65] J.A. Robinson, *A machine-oriented logic based on resolution principle*, Journal of the ACM, Vol. 12, pp. 23-41, 1965.

[Rob65b] J.A. Robinson, *Automatic deduction with hyper-resolution*, International Journal of Computer and Mathematics, Vol. 1, pp. 227-234, 1965.

[Rob67] J.A. Robinson, *A review of automatic theorem proving*, Proceedings of The Symposium in Appl. Math., 1967.

[Rob67b] J.A. Robinson, *Heuristic and complete processes in the mechanization of theorem proving*, Systems and Computer Science, pp. 116-124, University of Toronto Press, Toronto, 1967.

[Rob68] J.A. Robinson, *The generalized resolution principle*, Machine Intelligence, Vol. 3, pp. 77-94, 1968.

[Sch97] Johann Schumann, *SiCoTHEO-simple competitive parallel theorem provers based on SETHEO*, Parallel Processing for Artificial Intelligence, Vol. 3, pp. 231-245, 1997.

[SL90] Johann Schumann and R. Letz, *PARTHEO: a high performance parallel theorem prover*, Proceedings of CADE-10, LNAI, 449, pp. 40-56, 1990.

[SS98] Geoff Sutcliffe, C. B. Suttner, *The TPTP problem library*, Journal of Automated Reasoning, Vol. 21, pp. 177-203, 1998.

[SS98b] C. B. Suttner, Geoff Sutcliffe, *The CADE-14 ATP system competition*, Journal of Automated Reasoning, Vol. 21, pp. 99-134, 1998.

[SS01] Geoff Sutcliffe and C. B. Suttner, *Evaluating general purpose automated theorem proving systems*, Artificial Intelligence, Vol. 131, pp. 39-54, 2001.

[Sti88] Mark E. Stickel, *A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler*, Journal of Automated Reasoning, Vol. 4, pp. 354-381, 1988.

[Sut92] Geoff Sutcliffe, *A heterogeneous parallel deduction system*, Proceedings of the FGCS Workshop on automated deduction: logic programming and parallel computing approaches, pp. 5-13, 1992.

[Sut97] Geoff Sutcliffe, *http://www.cs.miami.edu/~tptp/CASC/14/Call.html*

[Sut99] C. B. Suttner, *SPS-Parallelism + SETHEO = SPTHEO*, Journal of Automated Reasoning, Vol. 22, pp. 397-431, 1999.

[SW00] Gernot Stenz and Andreas Wolf, *Scheduling methods for parallel automated theorem proving*, Canadian AI 2000, LNAI, 1822, pp. 254-266, 2000.

[Urq87] A. Urquhart, *Hard examples for resolution*, Journal of the ACM, Vol. 34, pp. 209-219, 1987.

[Wan85] Tie Cheng Wang, *Designing examples for semantically guided hierarchical deduction*, Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1201-1207, Los Angeles, CA., 1985.

[WCR64] L. Wos, D. Carson and G. Robinson, *The unit preference strategy in theorem proving*, Proceedings of AFIPS 1964 Fall Joint Comput. Conf., Vol. 26, pp. 615-621.

[WL99] S. Wolf and R. Letz, *Strategy parallelism in automated theorem proving*, International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI), 13(2): 219-245, 1999.

[WL01] Chin-hung Wu and S.J. Lee, *Parallelization of a hyper-linking-based theorem prover*, Journal of Automated Reasoning, Vol. 26, pp. 67-106, 2001.

[WM76] G. A. Wilson and J. Minker, *Resolution,refinements and search strategies: a comparative study*, IEEE Transactions on Computer, C-25, pp. 782-801, 1976.

[Wos88] L. Wos, *Automated reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[WOLB92] L. Wos, R. Overbeek, E. Lusk and J. Boyle, *Automated reasoning: Introduction and Application*, 2nd Edition, McGraw-Hill Inc., NY, 1992.

[WRC65] L. Wos, G. Robinson and D. Carson, *Efficiency and completeness of the set of support strategy in theorem proving*, Journal of the ACM, Vol. 12, Oct., pp. 536-541, 1965.

[YAN98] Q. Yu, M. Almulla, and M. Newborn, *Heuristics used by HERBY for semantic tree theorem proving*, Annals of Mathematics and Artificial Intelligence, Vol. 23, pp. 247-266, 1998.

[ZBH96] Hantao Zhang, Maria Paola Bonacina and Jieh Hsiang, *PSATO: A distributed propositional prover and its application to quasigroup problems*, J. Symbolic Computation, Vol. 11, pp. 1-18, 1996.

Appendix A

Experimental results

A.1 HERBY versus PrHERBY(30)

The table in this appendix lists the results of experiments on HERBY and PrHERBY (30 machines) with the theorems of CADE-14, MIX division.

The first and second columns show the sequential numbers and theorem names, respectively. The `Prf` columns indicate whether the theorem is solved('O') or not('-'). The `CPU` columns give the CPU time. The `WC` columns give the wall clock time. The `SYS` column gives the system time taken for the master to implement the parallel strategy. When a slave finds a proof, the system time is negligible and is indicated by '-'.

The `RES` column contains the number of resolutions generated until a proof is obtained or the time limit is reached. For PrHERBY, it is the resolutions generated by the slave that found the proof if there is a proof. Otherwise, it is the ones generated by the master. The number of resolutions is equivalent to the number of clauses produced.

The `Node` columns give the number of nodes generated until a proof is obtained or the time limit is reached. For PrHERBY, it is the nodes generated by the slave that found the proof if there is a proof. Otherwise, it is the ones generated by the master using IDDFS.

The `Atom` column contains the number of atoms generated by the master until a proof is obtained or the time limit is reached. The `UA` column contains the number of atoms distributed by the master to slaves. This number cannot exceed the number of atoms in the `Atom` column. The `SA` column gives the number of atoms generated by the slave that found a proof. A 'NA' in the `SA` column indicates that the master itself found the proof.

Table A.1: HERBY vs. PrHERBY(30) : test results

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
1	GRP048-2	O	121.23	123	334	62	O	0.42	-	0	1831	92	2026	217	19
2	LCL003-1	-	297.87	300	723	341	-	271.27	23.29	299	4559626	1749505	291031	5584	
3	LCL004-1	-	296.85	300	754	351	-	265.55	28.98	299	5718803	2328065	262316	5935	
4	LCL006-1	-	305.91	306	233	99	O	57.47	8.52	66	1848666	644968	121883	1118	NA
5	LCL009-1	-	298.81	300	313	121	O	17.76	2.94	21	503820	215469	37760	644	NA
6	LCL010-1	-	298.94	300	450	146	O	0.27	0.05	0	4728	2052	655	133	NA
7	LCL011-1	-	303.72	304	250	102	O	5.41	0.99	7	162635	70749	13908	532	NA
8	LCL012-1	-	298.12	300	427	206	-	258.90	38.20	299	8586406	3438081	95967	2112	
9	LCL014-1	-	304.16	304	216	99	-	257.20	39.62	299	8949401	3535361	109112	2312	
10	LCL015-1	-	298.25	300	467	199	-	259.05	36.61	299	8757643	3315713	37183	3903	
11	LCL016-1	-	299.13	300	307	133	-	257.38	37.97	299	8507239	3490817	27891	3598	
12	LCL017-1	-	64.27	66	861	398	-	261.59	34.75	299	8296743	3153921	59772	2255	
13	LCL018-1	-	27.36	28	778	398	-	258.83	37.56	299	8899194	3358209	26248	2699	
14	LCL022-1	-	298.65	300	313	121	O	9.15	1.55	11	272434	119622	23929	577	NA
15	LCL023-1	-	305.51	306	331	126	O	17.85	2.77	21	529893	232180	39311	463	NA
16	LCL024-1	-	298.98	300	184	80	-	254.72	42.61	299	10551036	3972097	52240	1454	
17	LCL025-1	-	299.59	300	147	66	O	0.86	-	1	338	38	2951	288	23
18	LCL026-1	-	302.20	302	160	72	-	267.66	29.64	299	7515986	2533377	183142	1086	
19	LCL029-1	-	298.71	300	159	73	O	86.15	15.06	101	4179814	1245655	118125	885	NA
20	LCL030-1	-	299.56	300	137	63	-	282.22	15.48	299	4589420	1249281	162434	1096	
21	LCL033-1	-	306.50	307	329	155	O	0.09	0.04	0	3138	1378	158	15	NA
22	LCL034-1	-	297.38	300	423	205	-	271.22	25.51	299	6041226	2292225	128597	1594	
23	LCL036-1	-	298.86	300	287	136	-	259.04	37.97	299	9631845	3388929	183622	1849	
24	LCL039-1	-	299.59	300	123	54	-	247.25	50.59	299	13781191	4283905	414394	758	
25	LCL040-1	-	300.65	301	131	55	-	275.95	21.52	299	6639503	1725953	264954	879	
26	LCL042-1	-	305.03	305	323	164	-	281.73	16.28	299	5342735	1219585	286915	938	
27	LCL045-1	-	299.14	300	146	66	O	23.02	-	23	604	70	54597	356	39
28	LCL047-1	-	299.70	300	181	74	-	243.27	54.25	299	13101658	4865025	162844	1220	
29	LCL048-1	-	299.66	300	201	76	-	265.20	31.73	299	7440721	2688001	227146	1366	
30	LCL049-1	-	299.58	300	185	75	-	249.41	47.35	299	11543018	4218881	205218	1332	
31	LCL050-1	-	299.59	300	171	76	-	271.14	26.56	299	6491842	2313217	200623	1387	
32	LCL051-1	-	299.52	300	169	73	-	269.33	28.24	299	6581462	2369025	232601	1497	
33	LCL052-1	-	299.53	300	183	74	-	248.01	49.60	299	11973121	4399617	198896	1336	
34	LCL053-1	-	301.76	302	222	84	-	251.38	45.37	299	10539724	3895809	291465	1379	
35	LCL055-1	-	299.63	300	167	68	-	269.00	28.36	299	6807566	2424833	225262	1479	
36	LCL056-1	-	299.45	300	153	62	-	237.95	59.16	299	14511912	5467137	61054	280	
37	LCL057-1	-	299.50	300	197	75	-	237.05	60.20	299	15011375	5527553	174784	1339	
38	LCL058-1	-	299.60	300	187	75	-	252.52	44.78	299	10960105	4009985	187007	561	
39	LCL059-1	-	299.64	300	179	73	-	249.85	46.98	299	10994036	4045313	257752	1359	
40	LCL060-1	-	299.95	300	179	73	-	256.04	41.26	299	9900168	3520513	296043	1473	

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
41	LCL064-1	-	298.60	300	295	126	O	0.75	-	1	397	47	2992	342	29
42	LCL067-1	-	301.17	301	251	128	-	276.26	19.56	299	5291094	1687553	125685	364	
43	LCL068-1	-	298.47	300	317	138	-	246.80	50.34	299	12974805	4379649	300579	1606	
44	LCL070-1	-	299.99	300	209	90	-	278.02	18.74	299	4778464	1470465	216568	1678	
45	LCL071-1	-	299.43	301	387	174	-	250.85	46.21	299	12490789	3781121	354202	1747	
46	LCL075-1	-	297.48	300	614	286	O	17.98	2.66	21	542550	191984	40086	1404	NA
47	LCL080-1	-	299.33	300	180	82	-	283.08	14.97	299	4052599	1188865	193636	1080	
48	LCL083-1	-	298.07	300	693	338	O	69.41	17.09	87	3867130	1491587	67839	1214	NA
49	LCL086-1	-	302.62	303	2427	146	O	3.52	1.16	5	220223	85514	13298	684	NA
50	LCL087-1	-	299.01	300	259	121	O	0.95	0.33	2	40667	15954	3452	380	NA
51	LCL088-1	-	310.22	310	807	179	O	51.30	8.12	61	1996955	696010	58940	1439	NA
52	LCL089-1	-	299.88	300	425	210	-	267.50	29.59	299	6249822	2521601	153612	1963	
53	LCL090-1	-	300.88	303	560	262	-	246.11	49.91	299	12633823	4209153	328705	2884	
54	LCL091-1	-	158.66	160	865	398	-	267.95	28.83	299	8064365	2485761	219086	2554	
55	LCL092-1	-	297.80	300	694	320	-	270.08	26.78	299	7202183	2196993	245851	2687	
56	LCL093-1	-	164.08	165	864	398	-	279.71	16.94	299	4171609	1316865	191155	2649	
57	LCL094-1	-	300.59	302	264	120	-	254.98	41.39	299	10168643	3693569	166817	2253	
58	LCL095-1	-	297.73	300	751	365	-	263.71	33.01	299	7797425	2727425	248256	2849	
59	LCL101-1	-	303.52	304	244	122	O	0.91	0.11	1	17666	7654	1763	401	NA
60	LCL102-1	-	298.86	300	171	75	O	111.73	4.13	116	858593	281333	75936	1663	NA
61	LCL103-1	-	511.69	512	11108	128	-	281.43	15.33	299	3266652	1285633	89297	450	
62	LCL104-1	-	309.86	310	205	93	O	1.51	0.19	2	20926	8129	2922	508	NA
63	LCL107-1	-	301.30	303	350	168	O	0.20	0.00	0	1210	820	103	21	NA
64	LCL108-1	-	298.10	300	418	205	O	5.13	0.42	6	43021	14981	6670	692	NA
65	LCL110-1	-	299.57	300	151	68	O	8.41	-	8	634	82	31266	560	43
66	LCL111-1	-	299.51	300	123	57	O	0.30	0.10	1	11725	3653	678	0	NA
67	LCL112-1	-	299.50	300	125	58	-	243.52	53.57	299	15111158	4701185	312312	1149	
68	LCL113-1	-	299.61	300	119	55	-	260.52	36.34	299	10689767	3057665	279465	1219	
69	LCL114-1	-	299.52	300	153	67	-	260.55	37.08	299	10867888	3061249	311622	1197	
70	LCL115-1	-	299.61	300	147	67	-	259.32	37.72	299	11369372	3260417	232231	1117	
71	LCL116-1	-	299.48	300	135	62	-	257.02	40.39	299	11537157	3327489	323947	1238	
72	LCL118-1	-	299.24	300	370	175	O	2.88	0.66	4	96981	48591	6890	698	NA
73	LCL120-1	-	36.67	38	789	398	O	3.30	0.78	4	120688	50089	9009	617	NA
74	LCL121-1	-	298.16	300	303	146	-	267.25	29.84	299	6980937	2607105	122833	2019	
75	LCL123-1	-	298.96	300	279	134	O	260.26	18.34	280	4128455	1559048	101397	2166	NA
76	LCL127-1	-	298.26	300	769	366	-	279.29	16.02	299	3408373	1397761	60651	2358	
77	LCL128-1	-	302.57	303	280	134	-	278.45	18.32	299	4052916	1585665	73627	1881	
78	LCL130-1	-	297.82	300	367	176	O	0.05	0.01	0	673	324	51	0	NA
79	LCL131-1	-	301.27	302	331	164	-	274.55	18.76	299	4549904	1675777	88868	1715	
80	LCL182-1	-	299.04	300	289	129	O	1.56	-	1	584	61	9552	439	32
81	LCL187-1	O	0.00	0	5	2	O	0.12	-	0	35	5	0	0	2
82	LCL192-1	O	0.01	0	9	3	O	0.10	-	0	73	13	0	0	4
83	LCL194-1	O	0.01	0	9	3	O	0.13	-	0	80	13	0	0	4
84	LCL195-1	O	98.25	98	245	97	O	0.34	-	1	828	87	1246	138	16
85	LCL196-1	-	299.27	300	266	118	-	261.33	35.15	299	10706189	2659329	510346	2153	
86	LCL198-1	-	299.02	300	265	118	-	260.49	36.57	299	11180372	2886657	535771	2218	
87	LCL201-1	-	299.09	300	287	127	O	6.56	1.27	8	299621	70691	34606	874	NA
88	LCL204-1	-	303.63	304	311	137	O	0.12	-	0	242	31	770	93	13
89	LCL207-1	O	85.89	86	416	99	O	0.12	-	0	175	22	615	67	10
90	LCL208-1	O	24.87	25	169	67	O	0.20	-	0	255	33	808	97	16

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
91	LCL210-1	-	300.59	301	289	128	O	0.00	-	0	237	28	758	164	10
92	LCL211-1	O	0.01	0	11	4	O	0.11	-	0	109	15	362	6	5
93	LCL213-1	O	0.27	0	51	19	O	0.20	-	0	158	19	832	79	9
94	LCL214-1	O	0.16	0	41	16	O	0.12	-	0	124	18	472	40	8
95	LCL215-1	O	0.26	0	51	19	O	0.17	-	0	237	23	1086	142	12
96	LCL216-1	O	28.17	28	280	67	O	0.18	-	0	139	19	592	48	9
97	LCL217-1	O	25.62	25	276	67	O	0.11	-	0	157	19	1059	87	9
98	LCL218-1	O	97.38	98	242	97	O	0.16	-	0	330	40	994	133	13
99	LCL224-1	-	303.03	304	293	130	-	274.08	23.02	299	7258510	1730049	368212	1612	
100	LCL230-1	O	117.74	118	268	108	O	0.12	-	0	236	28	1051	111	11
101	LCL231-1	-	299.16	300	303	133	O	0.38	-	0	532	57	1502	167	17
102	LCL256-1	-	299.80	300	181	73	-	236.08	61.41	299	15142023	5686785	143295	1177	
103	NUM002-1	O	0.01	0	15	5	O	0.12	-	0	143	14	0	0	5
104	NUM003-1	O	0.17	0	22	9	O	0.15	-	0	261	23	280	27	7
105	NUM004-1	O	0.01	0	7	3	O	0.11	-	0	81	9	0	0	4
106	PLA004-1	-	298.86	300	177	76	-	263.89	33.60	299	13202915	1925121	927598	870	
107	PLA004-2	-	299.50	300	191	84	-	264.16	33.12	299	12528421	1933313	839214	894	
108	PLA005-1	-	299.44	300	185	81	-	263.13	33.43	299	13233359	1923073	918440	879	
109	PLA005-2	-	299.51	300	177	76	-	265.56	31.95	299	12615378	1925633	850023	889	
110	PLA007-1	-	299.60	300	159	70	-	252.96	44.30	299	9614769	3182081	772496	784	
111	PLA008-1	-	299.48	300	191	84	-	263.77	33.55	299	12919301	1957889	877135	888	
112	PLA009-1	-	299.56	300	183	79	-	264.92	32.71	299	9886600	2019841	827129	849	
113	PLA009-2	-	299.44	300	191	84	-	265.12	32.59	299	9731329	2011649	781410	865	
114	PLA010-1	-	299.34	300	177	76	-	265.30	32.13	299	12403650	1934337	832606	892	
115	PLA011-1	-	299.52	300	171	73	-	265.48	31.91	299	13279368	1911297	875008	871	
116	PLA011-2	-	299.52	300	185	81	-	265.55	32.15	299	11409772	1944577	798324	821	
117	PLA012-1	-	299.56	300	177	77	-	264.15	32.87	299	13837951	2001921	918513	867	
118	PLA013-1	-	299.52	300	181	79	-	264.16	33.20	299	13045213	2001921	846438	892	
119	PLA014-1	-	299.46	300	177	77	-	264.61	32.11	299	13299972	1931777	862688	867	
120	PLA014-2	-	299.51	300	177	76	-	267.11	30.34	299	10951544	1844737	744254	722	
121	PLA015-1	-	299.45	300	185	81	-	263.85	33.84	299	13276267	1939969	930768	870	
122	PLA016-1	-	300.22	300	176	75	-	258.05	38.41	299	8134417	2637825	668090	876	
123	PLA018-1	-	298.73	300	189	83	-	263.66	32.92	299	12510835	1902593	846839	888	
124	PLA019-1	-	299.67	300	177	76	-	257.52	39.43	299	8363446	2707457	673292	869	
125	PLA021-1	-	299.45	300	177	76	-	262.87	34.80	299	12263421	2102785	889131	961	
126	PLA022-1	O	2.46	2	1898	52	-	262.23	34.58	299	12591456	2064385	952165	943	
127	PLA022-2	O	3.95	4	132	51	-	261.20	35.84	299	14339879	2134017	964367	933	
128	PLA023-1	-	298.89	300	171	73	-	261.77	35.11	299	14265705	2117633	954280	919	
129	BOO004-1	O	1.35	1	66	26	O	1.08	-	1	1862	30	10790	254	21
130	BOO007-1	-	297.64	300	179	76	-	218.44	78.87	299	15867830	6948865	448199	1424	
131	BOO008-1	-	299.96	301	183	78	-	219.70	77.81	299	15977998	6912001	424276	1367	
132	BOO009-1	O	1.01	1	72	26	O	0.87	-	1	2733	63	15051	392	28
133	BOO010-1	-	299.01	300	150	62	O	1.07	-	1	3963	75	19110	449	28
134	BOO012-1	-	299.61	300	153	60	O	0.91	-	0	2205	39	16571	327	23
135	BOO014-1	-	297.79	300	175	75	-	218.45	79.35	299	16380115	6955009	492932	1000	
136	BOO015-1	-	301.18	302	168	72	-	214.53	83.11	299	16971928	7128577	456809	1372	
137	BOO016-1	-	298.09	300	170	72	-	220.77	76.51	299	15860579	6610433	452440	1243	
138	BOO017-1	-	298.50	300	165	70	-	220.76	76.09	299	15833829	6600193	481138	1302	
139	CAT001-1	-	298.73	300	337	145	-	224.86	72.08	299	16289640	6489089	196937	3298	
140	CAT001-4	-	299.20	300	771	269	O	1.01	-	1	299	33	12255	364	21

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
141	CAT002-1	-	298.24	300	361	156	-	228.73	68.29	299	16080624	6184449	168507	2516	
142	CAT002-4	-	299.11	300	333	150	O	0.66	0.22	1	51470	17789	1726	0	NA
143	CAT003-1	-	298.57	300	4771	165	O	103.59	-	106	22240	868	112162	2164	139
144	CAT003-2	-	299.34	300	308	130	O	73.15	-	74	688	87	216081	1311	65
145	CAT003-4	O	0.00	0	17	5	O	0.23	-	0	181	21	0	0	6
146	CAT004-1	-	298.53	300	293	127	O	165.48	-	168	71054	3306	145833	2875	154
147	CAT004-4	-	299.16	300	323	147	O	0.91	-	1	1040	104	10599	370	30
148	CAT005-4	O	0.18	0	81	30	O	0.52	-	0	2759	338	11203	393	31
149	CAT006-4	O	70.99	71	7845	117	O	1.62	-	2	1804	227	12845	609	42
150	CAT009-1	-	298.05	300	262	120	O	0.34	-	0	679	44	3621	223	30
151	CAT009-4	-	299.20	300	345	145	O	0.65	-	1	1080	144	18095	483	37
152	CAT010-1	-	298.77	300	261	115	O	0.18	-	0	427	32	1510	100	13
153	CAT010-4	-	299.08	300	341	148	-	231.00	66.10	299	17854728	5576193	599313	2454	
154	CAT011-4	O	23.69	23	192	73	O	119.58	-	120	14331	1833	170945	1238	152
155	CAT014-4	O	5.53	5	134	52	O	7.56	-	8	2255	267	39853	908	63
156	CAT018-1	O	0.09	0	45	18	O	0.21	-	0	488	38	117	12	17
157	CAT018-4	-	298.48	300	367	157	-	227.56	69.73	299	18098163	5958145	562419	1886	
158	CID001-1	-	299.39	300	253	108	-	227.98	69.51	299	16309993	5759489	651207	1347	
159	CID003-2	-	299.48	300	71	35	-	277.70	20.18	299	7909180	1526273	288136	662	
160	CIV001-1	-	299.49	300	221	90	-	293.28	4.48	299	1856219	239105	140579	844	
161	COL002-3	-	299.52	300	218	98	O	1.16	0.05	1	12573	2693	886	0	NA
162	COL003-3	-	299.30	300	297	130	-	264.25	33.05	299	7158119	2627585	361738	2284	
163	COL003-4	-	299.88	300	314	131	-	264.48	32.50	299	7151984	2627585	364603	2296	
164	COL003-5	-	298.61	300	357	149	-	265.04	32.10	299	7238007	2662913	363255	2363	
165	COL003-6	-	300.70	301	300	129	-	264.95	32.19	299	7181374	2634753	365693	2226	
166	COL003-7	-	299.05	300	327	138	-	271.27	25.97	299	5957051	1994241	437931	2368	
167	COL003-8	-	300.40	301	361	153	-	271.28	25.89	299	5948057	1992193	434451	2453	
168	COL003-9	-	299.30	300	306	129	-	270.11	27.14	299	6147156	2053121	438184	2255	
169	COL042-2	-	299.37	300	329	139	-	271.39	25.63	299	5654281	1891329	430844	2235	
170	COL042-3	-	299.52	300	287	122	-	271.43	25.80	299	5880563	1968641	434666	2310	
171	COL042-4	-	299.52	300	289	123	-	271.93	25.20	299	5644628	1889281	433438	2330	
172	GRP012-3	O	0.22	0	41	16	O	0.70	-	0	1332	73	10043	217	20
173	GRP051-1	-	298.99	300	245	122	-	287.43	9.54	299	1645797	801793	63245	2330	
174	GRP052-1	-	299.54	300	171	85	-	287.86	9.18	299	1538337	770561	67638	2530	
175	GRP053-1	-	299.38	300	219	109	-	287.94	9.45	299	1598814	790017	58229	2517	
176	GRP056-1	-	299.38	300	249	124	-	287.30	10.05	299	1654898	817153	58020	2424	
177	GRP057-1	-	299.49	300	193	96	-	290.55	6.91	299	1139220	598017	62249	2223	
178	GRP072-1	-	299.34	300	201	100	-	289.95	6.95	299	1265869	514049	122400	1368	
179	GRP074-1	-	299.33	300	174	87	-	290.57	7.26	299	1267852	516097	119364	1189	
180	GRP075-1	-	299.40	300	213	103	-	289.77	7.20	299	1329367	483841	149898	1496	
181	GRP076-1	-	299.37	300	217	105	-	289.45	8.27	299	1547586	563713	166860	1113	
182	GRP077-1	-	299.45	300	199	98	-	289.90	7.75	299	1572787	559105	163411	1421	
183	GRP078-1	-	299.38	300	216	105	-	290.35	7.35	299	1543777	556033	169115	1427	
184	GRP079-1	-	299.46	300	199	98	-	290.03	7.67	299	1594160	571905	166092	1383	
185	GRP080-1	-	299.37	300	213	103	-	289.48	8.29	299	1551648	565249	168723	1448	
186	GRP085-1	-	299.34	300	210	105	-	281.70	15.80	299	3153278	1441793	47088	1563	
187	GRP086-1	-	299.45	300	175	87	-	281.56	15.91	299	3107788	1432577	44395	1362	
188	GRP087-1	-	299.46	300	194	97	-	281.62	15.73	299	3134697	1450497	45889	1383	
189	GRP097-1	-	299.48	300	163	81	-	285.16	12.60	299	2624965	1035777	113945	1257	
190	GRP099-1	-	299.40	300	199	98	-	285.47	12.02	299	2516578	909313	185671	1098	

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
191	GRP100-1	-	299.41	300	199	98	-	285.09	11.91	299	2505570	905729	187958	1356	
192	GRP101-1	-	299.36	300	217	105	-	285.30	12.55	299	2525335	920065	186696	1085	
193	GRP102-1	-	299.44	300	217	105	-	285.76	11.70	299	2494802	919041	192364	1398	
194	GRP103-1	-	299.52	300	217	105	-	285.84	12.06	299	2515870	924673	189007	1380	
195	GRP104-1	-	299.52	300	168	83	-	279.72	16.94	299	3408816	1450497	108369	1128	
196	GRP105-1	-	299.38	300	167	83	-	275.99	16.77	299	3360020	1427457	107641	1174	
197	GRP108-1	-	299.30	300	195	97	-	280.02	16.93	299	3526833	1504769	105489	1259	
198	GRP109-1	-	299.42	300	173	86	-	279.98	16.83	299	3409194	1455105	107707	1207	
199	HEN003-1	-	298.79	300	157	72	-	190.18	106.30	299	15352522	9968129	46812	1940	
200	HEN003-3	O	99.73	100	233	103	O	0.21	-	0	281	42	571	158	18
201	HEN004-1	-	299.98	301	124	56	-	188.00	108.72	299	15420317	10150913	41569	1582	
202	HEN004-3	-	298.85	300	276	130	-	225.68	70.55	299	14674233	6097409	345744	2410	
203	HEN005-1	-	299.19	301	261	111	O	226.30	-	227	18261	475	39514	2143	124
204	HEN005-3	-	298.99	300	307	138	O	0.29	-	0	503	84	895	269	29
205	HEN006-1	-	298.66	300	227	99	-	195.12	102.07	299	16437012	9500673	41811	1480	
206	HEN006-3	-	299.87	301	302	135	-	238.12	59.12	299	12535873	5172737	309612	2258	
207	HEN006-5	-	299.44	300	248	110	-	264.13	33.38	299	9319396	2683393	427134	1766	
208	HEN007-1	-	297.77	300	245	108	-	196.28	100.76	299	16454859	9431553	44755	1871	
209	HEN007-3	-	299.07	300	297	136	-	250.32	46.80	299	10443084	4180993	335281	2272	
210	HEN008-1	-	212.34	214	446	162	O	0.26	-	0	916	40	298	88	20
211	HEN008-3	O	32.66	33	376	86	O	0.05	-	0	142	25	100	30	11
212	HEN009-1	-	298.30	300	231	98	-	195.73	101.35	299	16928443	9478657	45786	2207	
213	HEN009-3	-	299.12	300	355	168	-	251.70	45.05	299	10715874	3796481	404836	2452	
214	HEN009-5	-	299.27	300	215	104	O	63.55	-	64	27284	2363	131506	960	121
215	HEN010-1	-	298.95	300	271	105	-	196.03	101.36	299	16874365	9595393	39969	1569	
216	HEN010-5	-	299.48	300	205	95	-	262.82	34.28	299	7751246	2613249	543084	1521	
217	HEN011-1	-	237.21	239	259	119	-	202.02	95.12	299	16648068	8890369	73345	2346	
218	HEN011-5	-	299.27	300	235	106	-	276.03	21.02	299	5837864	1391105	498409	1262	
219	HEN012-1	-	298.65	300	192	86	-	188.40	108.41	299	15439262	10196481	48807	2106	
220	HEN012-3	-	299.37	300	273	127	O	0.18	-	0	224	36	361	104	13
221	LAT005-5	-	298.88	300	233	105	-	229.66	67.83	299	14032812	5670913	658056	1445	
222	LAT005-6	-	297.91	300	243	109	-	234.59	62.91	299	15253354	5315073	583109	1359	
223	LCL145-1	-	299.77	300	236	103	-	263.90	33.23	299	8267976	2307585	712067	1406	
224	LCL146-1	-	298.73	300	238	104	-	263.91	33.55	299	8361764	2349569	727650	1384	
225	LDA003-1	O	56.37	56	4325	116	O	0.25	-	0	493	70	589	182	18
226	NUM017-2	-	298.45	300	401	108	-	219.80	77.55	299	16326557	7085569	196607	1871	
227	RNG004-1	-	299.28	300	166	68	-	218.87	78.25	299	16144154	6929409	409433	1333	
228	RNG006-3	O	64.96	65	1982	57	O	126.65	-	127	34957	337	221874	1162	93
229	RNG007-1	-	298.95	300	163	66	-	215.57	81.59	299	16935960	7022593	521840	1245	
230	RNG008-1	-	300.16	301	141	60	-	220.78	76.99	299	15901874	6719489	472333	1095	
231	RNG037-1	O	144.12	144	1744	73	O	1.30	-	2	2669	67	6454	411	27
232	RNG039-1	-	299.25	300	155	67	-	254.77	42.85	299	11970882	3188225	708261	1124	
233	ROB011-1	-	299.05	300	319	136	-	263.89	33.39	299	8172479	2408961	678704	1580	
234	ROB016-1	-	299.27	300	355	151	O	0.29	-	0	192	28	1092	182	18
235	ANA002-2	-	298.24	300	51867	253	-	236.05	60.25	299	16337122	5629953	2914	2300	
236	SET005-1	O	0.01	0	29	6	O	0.16	-	0	722	42	24	13	7
237	SET007-1	O	0.02	0	46	7	O	0.14	-	0	1749	63	20	15	9
238	SET011-1	O	0.02	0	33	11	O	0.14	-	0	369	27	14	11	10
239	SET012-1	-	299.30	300	126	58	O	3.14	-	3	741	70	8470	384	32
240	SET012-2	-	299.19	300	273	120	-	222.58	74.85	299	17439722	6866945	232603	1811	

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
241	SET013-1	-	299.63	300	137	61	O	72.34	25.80	98	5334819	2312359	138354	1049	NA
242	SET013-2	-	299.63	300	203	85	-	222.30	74.70	299	16795634	6798337	226410	2342	
243	SET014-2	O	0.07	0	33	13	O	0.10	-	0	109	9	71	18	4
244	SET015-1	-	299.42	300	137	61	O	60.78	23.00	84	4525896	2028972	122369	854	NA
245	SET015-2	-	299.52	300	181	79	-	222.70	73.96	299	16747537	6728193	208496	1935	
246	SET055-6	O	0.08	0	7	2	O	0.24	-	0	43	7	0	0	2
247	ALG001-3	-	298.97	300	189	83	-	288.58	9.52	299	5217581	658433	166274	517	
248	CAT001-3	-	300.48	301	705	260	O	0.27	-	1	273	25	564	122	14
249	CAT002-3	O	56.95	57	884	79	O	0.05	-	0	202	17	272	57	7
250	CAT003-3	O	0.45	1	76	29	O	0.05	-	0	221	19	171	38	7
251	CAT004-3	-	299.39	300	301	126	O	0.53	-	0	982	89	2116	303	27
252	CAT005-3	O	23.45	23	3263	77	O	2.55	-	3	3975	455	4336	380	46
253	CAT006-3	O	5.66	5	558	60	O	6.31	-	6	3041	274	11733	609	47
254	CAT009-3	-	298.31	300	311	133	-	242.68	53.90	299	17765849	4722177	339048	1986	
255	CAT011-3	O	55.68	56	240	93	O	2.52	-	3	1049	105	5010	564	52
256	CAT014-3	-	299.11	300	319	135	O	4.27	-	4	4330	412	5346	542	54
257	GEO001-1	-	299.29	300	442	102	O	65.88	-	66	41196	811	3786	676	66
258	GEO001-2	-	299.44	300	207	92	O	0.41	-	0	1690	73	180	103	16
259	GEO002-1	-	298.38	300	1016	92	-	222.45	74.76	299	9816164	7068161	17998	1150	
260	GEO002-2	-	298.93	300	157	78	O	59.81	20.63	81	2577632	1896726	10191	880	NA
261	GEO004-1	-	298.84	300	143	62	-	218.84	78.05	299	10449185	7276033	15572	1170	
262	GEO005-1	-	298.96	300	1629	102	O	149.46	-	150	126298	2017	3719	514	79
263	GEO006-1	-	298.94	300	163	70	O	275.19	-	276	3847	95	7102	682	53
264	GEO006-2	-	298.76	300	417	150	-	224.12	72.89	299	10228379	6849537	18589	728	
265	GEO010-1	-	298.70	300	10749	79	-	225.36	71.37	299	10287984	6679041	10123	1331	
266	GEO010-2	-	299.18	300	560	84	-	228.30	68.63	299	10400807	6454273	7548	1021	
267	GEO011-1	-	299.03	300	295	60	-	222.09	75.11	299	10523280	6957569	6962	737	
268	GEO012-1	-	298.84	300	343	103	-	230.53	66.94	299	8524132	6150657	12800	1035	
269	GEO013-1	-	298.65	300	235	109	-	231.52	65.67	299	8752338	6038017	15091	1519	
270	GEO025-2	-	299.28	300	5347	72	O	129.96	-	130	166010	3565	26188	1820	106
271	GEO026-2	-	300.21	301	213	89	O	0.30	-	0	1599	43	393	215	17
272	GEO027-2	-	297.86	300	186	81	-	214.37	82.49	299	11872093	7621633	26443	1211	
273	GEO030-2	-	271.88	274	282	116	O	0.73	-	0	2618	76	536	281	23
274	GEO036-2	O	110.56	110	1036	51	-	225.57	71.12	299	10478933	6555137	1756	110	
275	GEO037-2	-	299.94	301	164	78	O	68.40	-	69	15084	505	6116	410	58
276	GEO039-2	-	298.57	300	214	99	O	1.11	-	1	1876	90	1297	154	17
277	GEO040-2	O	0.03	0	23	8	O	0.21	-	0	490	26	105	46	9
278	GEO041-2	-	298.44	300	196	86	-	218.46	79.12	299	11422233	7422977	18645	1077	
279	GEO042-2	-	299.10	300	121	56	-	216.35	80.66	299	11399882	7467009	11672	1065	
280	GEO043-2	-	298.80	300	164	81	-	219.38	77.33	299	10968817	7256065	12183	1183	
281	GEO048-2	-	298.77	300	205	96	-	229.35	67.67	299	10559969	6332417	7968	1262	
282	GEO059-2	-	299.30	300	146	69	O	8.55	-	9	2210	100	6596	396	42
283	GEO064-2	-	298.85	300	214	90	-	220.11	77.30	299	11167944	7189505	10281	1195	
284	GEO065-2	-	298.65	300	201	94	-	219.65	77.38	299	11309824	7244801	12272	1243	
285	GEO066-2	-	298.78	300	205	96	-	220.08	76.84	299	11339891	7315969	12122	1377	
286	GEO067-2	-	298.86	300	233	92	-	229.01	68.89	299	10970643	6438913	5135	491	
287	GEO076-4	-	23.83	24	60	24	-	252.52	44.72	299	6848753	4220929	278	221	
288	GEO077-4	-	22.48	23	45	18	O	116.74	-	118	28395	70	897	407	33
289	GRP008-1	O	0.23	0	34	14	O	0.29	-	1	384	18	1207	115	10
290	GRP025-1	-	298.14	300	244	107	-	257.09	40.06	299	11837784	3245569	383066	477	

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
291	GRP025-2	-	299.21	300	223	97	-	260.75	36.84	299	10714425	2974209	465065	786	
292	GRP026-2	-	299.43	300	201	89	-	246.34	51.24	299	13835834	4476417	342399	1233	
293	GRP027-1	-	299.60	300	133	56	-	233.34	64.06	299	16111436	5646849	290556	904	
294	GRP039-1	-	298.99	300	223	91	-	231.28	65.78	299	14374697	5592065	467187	1101	
295	GRP039-4	-	299.27	300	215	91	-	227.06	70.08	299	15297216	6003201	516041	1188	
296	GRP040-3	-	299.18	300	157	67	-	240.49	56.68	299	13441078	4798977	465874	1263	
297	NUM009-1	O	0.08	0	13	4	O	0.39	-	0	449	13	0	0	4
298	NUM042-1	-	298.86	300	111	50	O	1.19	-	1	358	24	594	84	12
299	NUM046-1	-	299.29	300	117	50	-	281.34	16.61	299	8877014	1398785	122081	461	
300	NUM061-1	-	300.84	301	102	48	-	280.10	17.68	299	8609977	1471489	158777	319	
301	NUM065-1	-	302.76	304	112	51	-	274.78	22.46	299	5859759	1472001	474703	272	
302	NUM066-1	-	299.18	300	103	49	-	277.33	20.11	299	6392918	1268737	460477	264	
303	NUM136-1	-	299.20	300	115	53	-	276.48	21.06	299	8772223	1701377	199639	368	
304	NUM139-1	O	0.00	0	3	1	O	0.16	-	1	26	3	0	0	1
305	NUM141-1	-	299.34	300	109	52	-	279.56	18.05	299	9297377	1568257	142074	350	
306	NUM142-1	-	299.22	300	115	55	-	286.39	11.33	299	7988370	758273	212348	374	
307	NUM149-1	-	299.30	300	109	51	-	286.66	10.77	299	8092137	760833	206023	384	
308	NUM159-1	-	299.31	300	119	55	-	287.53	10.56	299	7923721	745473	210675	363	
309	NUM180-1	-	299.20	300	115	55	O	0.68	-	0	315	19	341	27	7
310	NUM183-1	-	299.45	300	107	47	O	1.33	-	2	617	29	1220	89	13
311	NUM190-1	-	305.99	306	69	31	-	289.68	8.40	299	6513867	527361	195820	182	
312	NUM201-1	-	298.69	300	107	51	-	280.14	17.45	299	9963956	1480705	97273	350	
313	NUM228-1	O	0.00	0	3	1	O	0.20	-	0	28	3	0	0	1
314	NUM232-1	-	299.20	300	103	48	-	283.75	14.03	299	6751554	1065985	162967	354	
315	NUM235-1	-	299.29	300	109	52	-	277.13	20.35	299	8774272	1556481	262710	348	
316	RNG040-1	O	0.02	0	19	7	O	0.16	-	0	432	14	260	51	9
317	RNG041-1	O	0.01	0	13	4	O	0.11	-	0	300	13	0	0	4
318	SET017-6	-	299.28	300	112	50	-	282.35	15.60	299	6921610	1151489	274732	371	
319	SET019-4	-	299.04	300	120	50	O	0.30	-	0	202	8	113	67	3
320	SET024-4	O	4.85	5	82	31	O	0.48	-	0	557	11	149	60	5
321	SET025-4	-	311.34	312	104	43	O	0.27	-	0	784	12	0	0	4
322	SET025-9	-	298.64	300	101	44	O	14.92	-	16	8163	32	10192	198	16
323	SET027-4	O	0.05	0	13	4	O	0.24	-	0	153	13	0	0	4
324	SET031-4	-	319.74	320	106	45	O	1.56	-	1	1277	28	874	172	13
325	SET041-4	-	299.35	300	107	47	-	258.02	39.70	299	8172170	3525633	54831	314	
326	SET050-6	O	0.01	0	9	4	O	0.17	-	0	79	9	0	0	4
327	SET051-6	O	0.01	0	7	3	O	0.16	-	0	66	7	0	0	3
328	SET061-6	-	299.33	300	109	47	-	282.68	15.33	299	7219470	1071105	336041	450	
329	SET062-6	-	299.52	300	139	59	O	0.43	-	0	348	25	426	58	9
330	SET063-6	O	0.23	0	28	10	O	0.69	-	1	437	36	616	124	14
331	SET064-6	O	0.21	0	27	10	O	21.95	-	22	587	36	26291	277	18
332	SET067-6	-	299.30	300	107	46	-	285.09	13.14	299	6817700	881665	316045	448	
333	SET068-6	-	299.01	300	115	50	-	284.70	13.23	299	6599319	858625	328271	440	
334	SET071-6	-	299.32	300	145	66	O	33.22	-	50	510	22	44594	256	12
335	SET072-6	-	299.50	300	117	52	-	281.44	16.00	299	6879649	1201153	267423	369	
336	SET073-6	-	299.49	300	103	46	-	283.05	14.86	299	6388045	1105409	226463	368	
337	SET074-6	-	299.44	300	111	49	-	282.73	15.01	299	6486813	1095169	226621	353	
338	SET075-6	-	299.40	300	113	52	-	291.24	6.68	299	3403347	423425	170195	302	
339	SET076-6	-	299.65	300	111	46	-	277.13	20.45	299	7711945	1421313	329141	433	
340	SET078-6	O	0.06	0	11	5	O	0.19	-	0	246	11	0	0	5

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
341	SET079-6	-	299.25	300	115	50	-	281.38	16.53	299	7471857	1179137	311576	312	
342	SET080-6	O	0.13	0	17	6	O	0.19	-	1	310	17	204	35	6
343	SET081-6	O	0.07	0	13	4	O	0.20	-	0	179	13	0	0	4
344	SET082-6	O	85.14	85	95	42	O	49.43	-	53	703	48	44280	363	35
345	SET083-6	O	61.49	62	98	40	O	1.27	-	2	759	40	2240	82	15
346	SET084-6	O	72.12	73	99	40	O	79.04	-	85	1648	59	83762	301	18
347	SET085-6	O	12.64	12	60	24	O	0.54	-	1	453	22	555	43	8
348	SET093-6	O	0.07	1	9	4	O	0.23	-	0	277	13	0	0	6
349	SET094-6	O	38.16	38	74	30	O	2.74	-	3	610	23	3390	59	7
350	SET095-6	-	300.40	301	121	53	O	104.33	-	106	918	36	129298	343	20
351	SET096-6	-	299.23	300	138	58	-	283.96	14.11	299	7835410	995841	287836	550	
352	SET101-6	-	301.26	301	105	45	O	0.48	-	0	493	26	617	96	13
353	SET102-6	O	7.15	8	58	24	O	0.24	-	0	226	12	0	0	4
354	SET108-6	O	0.00	0	5	2	O	0.16	-	0	152	5	0	0	2
355	SET117-6	O	0.02	0	7	3	O	0.17	-	0	164	9	0	0	4
356	SET124-6	-	299.28	300	107	44	O	127.90	-	143	1980	70	147161	377	29
357	SET125-6	-	299.67	300	115	46	O	14.80	-	15	1384	67	19952	237	20
358	SET130-6	-	299.54	300	103	43	-	282.70	14.51	299	6201891	1158657	199050	433	
359	SET138-6	-	299.66	300	111	44	-	270.14	27.30	299	9884951	2248193	269212	321	
360	SET144-6	-	300.07	300	158	71	-	284.74	12.52	299	7741494	861697	262673	556	
361	SET146-6	-	298.52	300	135	61	-	285.16	12.68	299	8028049	881153	281858	421	
362	SET147-6	-	299.21	300	137	64	-	285.67	12.05	299	7680208	836609	274348	528	
363	SET148-6	-	299.27	300	133	62	-	285.49	12.22	299	7670649	833537	270904	538	
364	SET149-6	-	299.34	300	141	66	-	285.86	12.24	299	7544538	823809	268773	534	
365	SET151-6	-	300.83	301	136	61	-	285.16	12.87	299	7801076	842753	259898	522	
366	SET153-6	O	8.64	9	41	17	O	17.70	-	18	290	21	19601	129	15
367	SET163-6	-	304.92	305	156	69	-	279.16	12.61	299	7513588	836097	270119	506	
368	SET166-6	-	299.55	300	133	56	-	226.57	15.47	299	6909677	1166337	223955	557	
369	SET167-6	O	4.91	5	52	20	O	3.84	-	4	583	31	5834	185	12
370	SET168-6	O	3.60	3	34	13	O	0.56	-	0	320	16	590	40	6
371	SET183-6	-	303.91	305	151	67	-	254.74	11.78	299	7394195	829441	222881	544	
372	SET186-6	-	300.80	301	110	50	-	274.69	11.19	299	6469205	690177	268402	271	
373	SET187-6	O	148.41	148	125	53	O	1.05	-	1	256	24	613	117	16
374	SET188-6	-	298.59	300	121	56	-	251.21	11.61	299	6634602	734209	275185	251	
375	SET189-6	-	299.30	300	129	59	-	271.64	12.85	299	7708114	880129	274261	323	
376	SET192-6	O	3.61	3	57	24	O	0.59	-	0	240	21	235	30	11
377	SET193-6	O	17.88	18	77	32	O	33.00	-	66	608	36	28027	357	20
378	SET194-6	O	39.62	39	94	38	-	276.93	11.91	299	7026586	784897	291440	470	
379	SET195-6	O	29.75	30	86	34	O	1.10	-	1	1095	60	1415	165	16
380	SET196-6	O	0.00	0	5	2	O	0.16	-	0	46	5	0	0	2
381	SET197-6	O	0.01	0	5	2	O	0.17	-	0	46	5	0	0	2
382	SET199-6	O	42.41	42	108	44	O	1.20	-	1	359	32	585	122	9
383	SET201-6	-	299.23	300	156	69	O	0.88	-	1	492	39	1055	133	14
384	SET203-6	O	10.64	11	45	18	O	0.52	-	0	383	25	513	40	7
385	SET204-6	O	0.00	0	5	2	O	0.16	-	0	49	5	0	0	2
386	SET230-6	-	303.28	304	119	54	-	267.18	12.48	299	7507370	832001	276108	283	
387	SET231-6	O	0.00	0	3	1	O	0.15	-	0	22	3	0	0	1
388	SET232-6	O	34.49	35	92	38	O	0.88	-	1	308	19	836	102	11
389	SET233-6	O	26.52	27	86	36	O	0.91	-	0	303	12	982	131	5
390	SET234-6	O	0.13	0	9	3	O	0.34	-	0	156	9	116	0	3

No	Theorem	HERBY					PrHERBY(30)								
		Prf	CPU	WC	Node	SA	Prf	CPU	SYS	WC	RES	Node	Atom	UA	SA
391	SET235-6	-	299.04	300	175	76	O	0.84	-	1	364	17	871	139	10
392	SET236-6	O	22.88	23	88	36	O	0.51	-	0	266	14	414	54	7
393	SET238-6	-	299.63	300	134	57	O	0.70	-	1	498	19	645	59	7
394	SET239-6	O	0.13	0	16	5	O	0.34	-	0	225	16	0	0	5
395	SET240-6	-	298.74	300	143	62	O	0.29	-	0	239	11	197	31	5
396	SET241-6	-	299.38	300	131	58	O	3.44	-	3	430	26	4575	162	17
397	SET242-6	O	0.02	0	7	3	O	0.19	-	0	144	7	0	0	3
398	SET243-6	-	299.06	300	131	60	-	272.71	8.77	299	5286959	597505	154162	476	
399	SET245-6	-	299.34	300	135	60	-	253.12	12.62	299	7387683	848897	252865	450	
400	SET252-6	O	4.08	4	55	23	O	0.37	-	0	258	19	322	36	7
401	SET253-6	O	3.28	3	40	17	O	0.49	-	0	274	21	457	49	9
402	SET261-6	-	298.59	300	137	63	-	252.38	11.92	299	7401598	845825	256407	423	
403	SET386-6	O	80.70	81	97	42	O	4.94	-	5	505	25	4491	57	8
404	SET411-6	-	301.10	301	153	65	O	3.66	-	4	571	35	5353	211	14
405	SET451-6	O	14.52	15	93	38	O	0.93	-	0	340	27	1266	130	15
406	SET454-6	-	298.70	300	149	65	-	257.26	11.35	299	6429846	750081	269922	477	
407	SET479-6	O	0.02	0	7	3	O	0.12	-	0	74	7	0	0	3
408	SET506-6	-	299.16	300	129	62	-	265.51	9.96	299	6531532	636929	262397	273	
409	SET507-6	-	299.36	300	124	57	O	37.46	-	38	769	35	93076	189	14
410	SET510-6	-	299.37	300	137	61	-	263.34	12.65	299	7391327	832001	265768	444	
411	SET516-6	-	305.23	305	124	57	-	271.91	11.27	299	6735000	683521	287338	260	
412	SET517-6	-	299.55	300	117	54	-	270.20	10.62	299	6521293	650241	271116	210	
413	SET553-6	O	56.65	56	104	43	O	0.50	-	0	277	21	451	43	9
414	SET558-6	-	304.01	304	133	60	O	43.78	-	88	1122	28	68132	323	12
415	SET559-6	-	300.47	301	135	59	-	251.17	12.46	299	6896158	820737	281948	419	
416	SET561-6	-	298.81	300	137	60	-	270.10	15.20	299	6558537	1104897	203613	423	
417	SET562-6	-	299.50	300	129	55	-	256.78	14.67	299	6471081	1158657	187300	437	
418	SET564-6	-	301.77	302	156	69	O	4.80	-	5	734	31	3465	158	18
419	SET565-6	-	298.86	300	138	62	-	246.88	9.85	299	5434942	676353	177233	394	
420	SET566-6	O	54.23	54	103	45	-	251.12	9.20	299	5005540	656385	176116	441	

Index