Guiding Linear Deductions with Semantics

Thesis submitted by
Marianne Elizabeth Brown BSc(Hons) *JCU*
December 2003

for the Degree of Master of Science
in the School of Information Technology
James Cook University

Statement on Access

I, the under-signed, the author of this work, understand that James Cook University will make this thesis available for use within the University Library and, via the Australian Digital Theses network, for use elsewhere.

I understand that, as an unpublished work, a thesis has significant protection under the Copyright Act and I do not wish to place any restriction on access to this thesis.

_____ _____

Signature Date

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

_____ _____

Signature Date

Electronic Copy

I, the undersigned, the author of this work, declare that the electronic copy of this thesis provided to the James Cook University Library is an accurate copy of the print thesis submitted, within the limits of the technology available.

_____ _____

Signature Date

Acknowledgements

I would like to thank Dr. Geoff Sutcliffe, who was my supervisor for the first three years, and Dr. Bruce Litow, who was my supervisor for the last three years. This thesis would not have been finished without their support. I would like to also acknowledge the support and motivation provided by the staff at the School of Information Technology including Dr. Greg Allen and Mrs. Beverley Frangos. To my fellow research students Alan McCabe, Chris Christensen and Jarrod Trevathan, thanks for the coffee and conversations. And finally to family and friends, who for the most part, tried to avoid the inevitable question, “So, how’s the thesis going?”

Abstract

Guidance is a central issue in Automatic Theorem Proving systems due to the enormity of the search space that these systems navigate. Semantic guidance uses semantic information to direct the path an ATP system takes through the search space. The use of semantic information is potentially more powerful than syntactic information for guidance. This research aimed to discover a method for incorporating semantic guidance into linear deduction systems, in particular model elimination based linear systems. This has been achieved. The GLiDeS pruning strategy is a simple strategy of restricting the model elimination deduction to one where all A-literals are false in the guiding model. This can be easily incorporated into any model elimination based prover. Evaluation of the GLiDeS strategy has shown that when "good guidance" has been achieved, the benefit of this guidance is significant. However attempts to develop a heuristic for predicting which model will provide "good guidance" has been largely unsuccessful.

Original Contributions

1. Developed novel strategy (GLiDeS) for applying semantic guidance to full linear deduction systems.
2. Shown that the new GLiDeS strategy is sound but incomplete.
3. Shown that GLiDeS is complete for a small group of problems termed *Semantic Horn* and that this result is essentially equivalent to renaming [Slagle, 1967].
4. Implemented system to demonstrate ease of including GLiDeS into an existing linear theorem proving system, PTTP.
5. Evaluated performance of the GLiDeS semantic guidance strategy and concluded that overall the GLiDeS strategy does not provide significant improvement to PTTP's performance.
6. It has been shown that when good guidance is achieved the improvement in performance is significant. GLiDeS dramatically reduces the amount of search space covered before a proof is found (as reflected by the number of inferences made). In the best case, PTTP covered on average 8 times the search space that GLiDeS covered (See NHN_SEQ Table 6.7).

Material from this thesis has appeared in the following publications:

- M. Brown. Selecting Semantics for Use with Semantic Pruning of Linear Deductions, In McKay,Bob and Slaney, J. editor, *AI 2002: Advances in Artifical Intelligence. 15th Australian Joint Intelligence Canberra, Australia, December 2002 Proceedings*, number 2557 in LNAI. Springer-Verlag, Berlin Heidelberg, Germany, 2002.
- M. Brown and G. Sutcliffe. PTTP+GLiDeS - Semantically Guided PTTP, In D. McAllester, editor, *Automated Deduction - CADE-17: 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000 Proceedings*, p 719, number 1831 in LNAI. Springer-Verlag, New York, USA, 2000.

- M. Brown and G. Sutcliffe. PTTP+GLiDeS: Guiding Linear Deductions with Semantics. In N. Foo, editor, *Advanced Topics in Artifical Intelligence: 12th Australian Joint Conference on Artificial Intelligence, AI'99*, number 1747 in LNAI, pages 244–254. Springer-Verlag, 1999.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Technical Notation

This thesis investigates the application of semantic guidance to linear deduction systems. Guidance is a central issue in Automatic Theorem Proving (ATP) systems due to the enormity of the search space that these systems navigate. Semantic guidance uses semantic information to direct the path an ATP system takes through the search space. This is of interest because the use of semantic information is potentially more powerful than syntactic information for guidance. This chapter discusses these issues and introduces the notation and terminology used in this thesis.

This chapter contains:

1. Background to ATP systems.
2. Motivation for semantic guidance.
3. The objectives of this research.
4. Notation and definitions of terminology.
5. An overview of the thesis contents.

1.1 Background

The field of Automated Theorem Proving (ATP) involves the solving of problems, using logical reasoning, mechanically. Applications of ATP include logic circuit validation, research in mathematics and formal logic, real-time systems control, program debugging and verification, and expert systems. Research carried out in the ATP field has also contributed to other fields, e.g. deductive databases.

Modern ATP research owes much to the work by J.A. Robinson who developed the resolution procedure in 1965 [33]. Prior to this, methods based on Herbrand's theorem had been used. One of these was the Davis-Putnam method [12]. Many refinements for resolution have been developed including Set of Support [45], hyperresolution [34], linear resolution [24, 27], and model elimination [22].

ATP systems take as input known facts (called axioms) and a statement of what is thought to be true (called a conjecture). This data is represented in some form of logic. There are ATP systems that deal with propositional, temporal and modal logics (to name a few) but this dissertation deals with ATP systems that work with first order logic (FOL) represented in *conjunctive normal form* (CNF). The proof method used is *proof by contradiction*. In a proof by contradiction, the conjecture is negated (i.e. assume the conjecture is FALSE) and a contradiction is searched for. If a contradiction is found then the assumption that the conjecture is FALSE is incorrect, and so the conjecture is proved.

The main problem with using FOL is that it is semi-decidable i.e. the proofs, and therefore the theorems, are recursively enumerable, but the non-theorems are not. So it is possible that the search for a proof may continue forever when no proof exists.

Chapter 2 contains a review of ATP development, with a focus on linear deduction systems. This may be skipped by readers familiar with the area.

1.2 The Need for Semantic Guidance

For even relatively simple ATP problems, the search space that needs to be traversed to find a proof is large. Therefore it is necessary to control the search in an intelligent manner. Search guidance can be provided in two different ways: search space pruning, where areas of the search space are excluded from the search path, and search space ordering, where preference is given to some areas of the search space but the entire search space may be searched if necessary. Guidance can use either syntactic information (some physical feature of the clauses or literals) or semantic information (some truth value interpretation of the clauses or literals). For example, the unit preference strategy, where preference is given to resolutions involving a clause containing a single literal, is an ordering strategy based on syntax. Model Resolution uses an interpretation to divide the clause set into two subsets and restricts resolutions to those that involve one clause from each set. This is a pruning strategy based on semantics.

Semantic guidance has been used with some forward chaining (see 1.4 for definition) techniques. For example, Model resolution is a semantic forward chaining technique. Little research has been carried out on the use of semantic guidance with backward chaining techniques, such as linear deduction. Chapter 3 contains a review of semantic guidance strategies employed by both forward and backward chaining methods.

1.3 Research Objectives

The main objective of this research is to investigate ways to incorporate semantic guidance into linear deductions systems. It was decided to attempt to develop a semantic guidance strategy that could be incorporated into an existing linear system with only minor modifications to the system. Research has focussed on producing a fully automatic system with no input required from the user of the system beyond the provision of the input clauses in the appropriate formats. This requires the generation of a model to provide the interpretation needed by the semantic system. The system developed by this research is named GLiDeS which is an acronym for **G**uiding **Li**near **De**ductions with **S**emantics

1.4 Notation and Terminology

1.4.1 Basic Terminology

- Constants and variables are *terms*.
- A f is a function symbol of arity n, and $t_1, t_2, ..., t_n$ are terms then $f(t_1, t_2, ..., t_n)$ is a term.
- A constant can be thought of as a function of arity 0.
- If p is a propositional letter, then it is an *atom*.
- If p is a predicate symbol of arity n, $n > 0$, and $t_1, t_2, ..., t_n$ are terms, then $p(t_1, t_2, ..., t_n)$ is an atom.
- A *literal* is an atom or the negation of an atom.
- A literal that is an unnegated atom is called a *positive literal* and a negated atom is a *negative literal*.
- A *clause* is a disjunction of literals.
- A *unit clause* is a clause consisting of only one literal.
- A *Horn* clause is a clause that contains at most one positive literal. Horn clauses are of interest as it means the original FOL clause was an implication with a single consequence. The statement $\sim a_1 \vee \sim a_2 \vee \ldots \vee \sim a_n \vee a_{n+1}$ is equivalent to $(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \rightarrow a_{n+1}$.
- A clause consisting of all negative literals is called a *negative clause*.
- A clause consisting of all positive literals is called a *positive clause*.
- The *empty clause* is an empty disjunction, and is unsatisfiable. The empty clause is symbolized by $\square$.
- A set of clauses is equivalent to their conjunction.

- Two literals g and f are said to be complementary if $f = \sim g$ or $g = \sim f$.
- Given two FO literals d and e, a unifier ,σ, is a substitution such that $d\sigma = e\sigma$. If such a σ exists, d and e are said to be unifiable.
- A unifier σ is said to be a *most general unifier* (MGU) if for all other unifiers σ_i there exists an substitution τ such that $\sigma = \tau\sigma_i$.
- A theorem proving strategy is said to be *complete* if it can be shown that provided a proof exists then the strategy will find one.
- A theorem proving strategy is said to be *sound* if it can be shown that if a proof is found then the proof is correct.

For readers unfamiliar with the areas of FOL and ATP, there are many excellent texts that can provide any background material not fully explained in this thesis. The texts by Duffy [13] and Fitting [14] are recommended.

Theorem proving strategies may be described as *forward chaining* or *backward chaining*. In forward chaining, a strategy starts with known information (axioms) and attempt to infer clauses that contradict the conjecture. If the analogy of a paper based maze is used, the forward chaining prover picks a starting point from many options and attempts to reach the end-goal. When a dead end is encountered, the strategy selects a different path and tries again. As any child knows, the easiest way to solve such a maze is to start at the goal and work backwards to the starting point. This an analogous to backward chaining. A backward chaining strategy starts with the conjecture, and attempts to infer clauses that contradict a known axiom. For a more formal definition, see [13].

1.4.2 Notation

The notation used for identifier naming throughout this thesis follows the format used in Prolog [35]. Variables are indicated by identifiers that start with capital letters, while functors and predicates have identifiers that start with lower-case letters. The standard symbols $\vee, \wedge, \sim, \rightarrow$ are used for the logical operators OR, AND, NOT and IMPLIES.

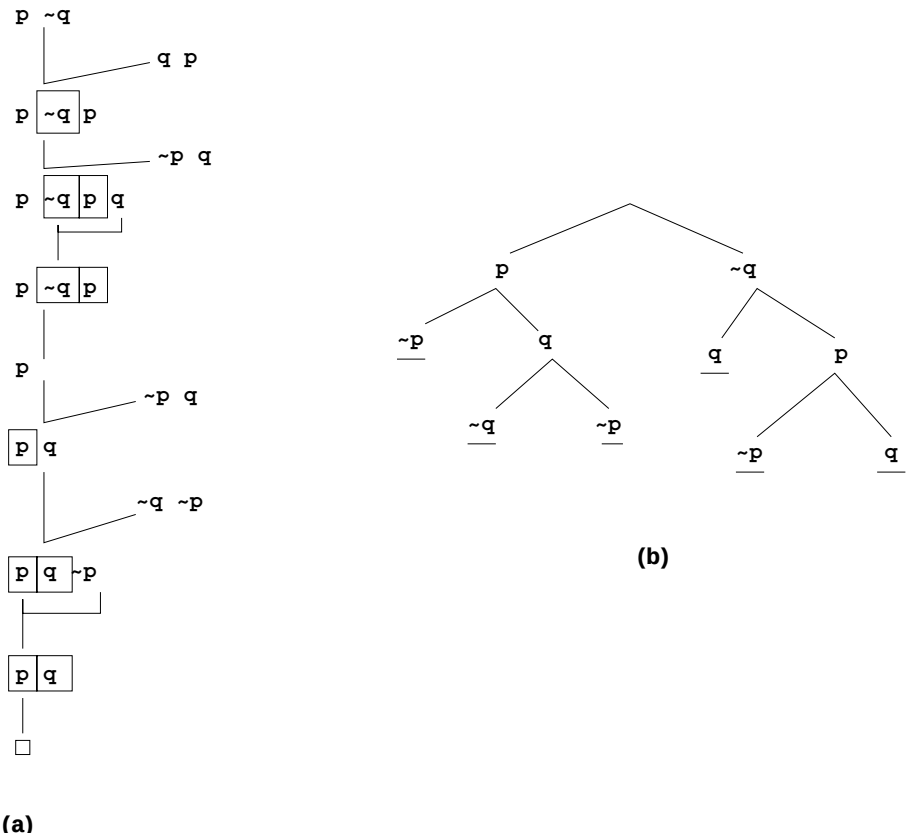

Figure 1.1: Two different visual representations of the same Model Elimination deductions: (a) traditional vertical representation, (b) tableau style representation.

Linear deductions can be visually represented in two different ways: i) as a vertical sequence of center clauses or ii) as a tableau. Figure 1.1 shows two different visual representations of the same model elimination deduction. Due to its more compact nature, the tableau representation will be used when displaying proofs in this thesis.

The tableau style representation (which shows the proof) should not be confused with the search space trees (which show all possible paths a deduction may take in the search for a proof) shown in Chapter 2 .

1.5 Thesis Structure

Chapter 2 contains a review of the development of linear deductions systems. Chapter 3 contains a review of the development of semantic guidance. Chapter 4 describes the GLiDeS pruning strategy and the underlying theory considerations. Chapter 5 describes a heuristic used to try and select the semantics to be used with GLiDeS. Chapter 6 contains the implementation and performance details for the GLiDeS system, with and without the model selection heuristic. Chapter 7 concludes.

Chapter 2

Linear Deduction Systems

In this chapter, the history of linear deduction is reviewed.

2.1 Ancient History

In 1965, Robinson's paper [33] on the resolution procedure changed the way automated theorem proving was approached. Until this pivotal paper, most of the research in this field was focused on instantiation based methods that involved determining the unsatisfiability status of ever expanding sets of ground clauses. The clauses are grounded by making variable substitution from the clause set's *Herbrand universe*. The Herbrand universe is the set of all terms that may be constructed from the function symbols in the clause set.

Subsets of the clause set's Herbrand universe are generated and used to ground the clauses. A check for unsatisfiability is then performed on these ground clauses. If the ground clauses were satisfiable, the next level Herbrand universe subset is generated and the process continues.

Example 1 *The Herbrand universe subsets are denoted by H_n where n is the level of the subset. So, given $T = \{\sim p(f(X)), p(Y) \vee \sim q(a), q(Z)\}$, the Herbrand universe subsets are: $H_0 = \{a\}, H_1 = \{a, f(a)\}, H_2 = \{a, f(a), f(f(a))\}$ and so on.*

In this example, only H_1 is needed to prove unsatisfiability giving $T_1 = \{\sim p(f(a)), p(f(a)) \vee$

$\sim q(a), q(a)\}$.

The resolution principle does not involve the grounding of clauses. Instead Robinson proposed that clauses be resolved together and the newly generated clause, called the resolvent, added to the clause set. For example, the ground clauses $\mathtt{a} \vee \mathtt{l}$ and $\mathtt{b} \vee \mathtt{l}'$ resolve upon the complementary literals $\mathtt{l}$ and $\mathtt{l}'$ to give the resolvent clause $\mathtt{a} \vee \mathtt{b}$. If the clauses are not ground then a substitution, σ, is made such that $\sigma\mathtt{l}$ and $\sigma\mathtt{l}'$ are complementary and the resolution takes place between $\sigma\mathtt{a} \vee \sigma\mathtt{l}$ and $\sigma\mathtt{b} \vee \sigma\mathtt{l}'$ giving the resolvent $\sigma\mathtt{a} \vee \sigma\mathtt{b}$. Clauses are resolved together until the empty clause is found.

The logic behind the resolution step is as follows: if $\mathtt{a} \vee \mathtt{l}$ and $\mathtt{b} \vee \mathtt{l}'$ are TRUE in some interpretation than so is any instance of them. However $\mathtt{l}$ and $\mathtt{l}'$ cannot both be TRUE as they contradict each other. So for both $\mathtt{a} \vee \mathtt{l}$ and $\mathtt{b} \vee \mathtt{l}'$ to be TRUE either

- $\mathtt{l}$ is FALSE (making $\mathtt{a}$ TRUE) and $\mathtt{l}'$ is TRUE or
- $\mathtt{l}'$ is FALSE (making $\mathtt{b}$ TRUE) and $\mathtt{l}$ is TRUE.

Either way, the resolvent $\mathtt{a} \vee \mathtt{b}$ must be TRUE. Thus, the resolvent is a logical consequence of the parent clauses. If the resolvent is found to be $\square$ then we have found the contradiction i.e. $\square$ is FALSE but if the parent clauses are TRUE then $\square$ must be TRUE.

One of Robinson's major contributions was the algorithm for finding an appropriate σ to use in the substitution process, or unification of the literals $\mathtt{L}$ and $\mathtt{L}'$. This algorithm, called the Unification Algorithm, finds the most general unifier. It was later modified by Paterson and Wegman [31] to include an occurs check, without which it is unsound.

The following algorithm is a simplified version of unification with occurs check as appeared in [13]:

Let E_1 and E_2 be two expressions with no variables in common. Let ϵ be the initial (empty) substitution. To unify E_1 and E_2 proceed as follows:

1. If E_1 and E_2 are identical then stop with the current substitution.

2. Let n be the leftmost position at which the symbols of E_1 and E_2 do not match, and let s_1 and s_2 be, respectively, the symbols at this position in E_1 and E_2. If neither s_1 nor s_2 is a variable then stop with failure. Otherwise, if s_1 is a variable then let it be x and let t be the term whose first symbol is s_2, else let s_2 (which must thus be a variable) be x and let t be the term whose first symbol is s_1.

3. If x occurs in t then stop with failure. Otherwise, apply the substitution $(x \rightarrow t)$ to E_1 and E_2, add $x \rightarrow t$ to the current substitution, and then return to step 1.

The resolution principle is much more efficient than Herbrand based methods, but can result in the generation of many unnecessary clauses (for example, tautologies or duplicate clauses). To apply the resolution principle in a systematic way, the simplest method is to perform level-saturation:

1. Let $k = 0$, $S_k = \{\}$, $R_k = S$ where S is the input clause set.

2. Let $R_{k+1} = \{\}$.

3. Compute all resolvents where one parent is from S_k, and one parent is from R_k. Place resolvents in R_{k+1}.

4. Compute all resolvents for pairs of clauses in S_k. Place resolvents in R_{k+1}.

5. Check R_{k+1} for the empty clause. If found exit.

6. Let $S_{k+1} = S_k \cup R_k$.

7. Let $k = k + 1$. Return to 2.

Using this method, clauses may be generated that do not contribute to the finding of a proof. But even with the removal of these obviously unnecessary clauses, resolution still produces many new clauses at each level. Much of the research on resolution based systems is focused on reducing the number of clauses produced at each level.

Example 2 *Given* $S = \{\mathtt{p} \vee \mathtt{q}, \sim\mathtt{p}, \sim\mathtt{q} \vee \mathtt{r}, \sim\mathtt{r} \vee \mathtt{s}, \sim\mathtt{r} \vee \sim\mathtt{s} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{t}\}$

Level Saturation Resolution - removing duplicate clauses and tautologies[1]*:*

$S_0 = \{\}\ R_0 = S$

$R_1 = \{\mathtt{q}, \mathtt{p} \vee \mathtt{r}, \sim\mathtt{t}, \sim\mathtt{q} \vee \mathtt{s}, \sim\mathtt{q} \vee \sim\mathtt{s} \vee \mathtt{t}, \sim\mathtt{r} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{r} \vee \sim\mathtt{s}\}$

Generating R_1 takes 7 inferences steps and results in 7 new clauses.

$S_1 = S_0 \cup R_0$

$R_2 = \{\mathtt{p} \vee \mathtt{s}, \mathtt{p} \vee \sim\mathtt{s} \vee \mathtt{t}, \mathtt{r}, \sim\mathtt{r} \vee \sim\mathtt{s}, \sout{\mathtt{r}}, \sim\mathtt{q} \vee \mathtt{t}, \sim\mathtt{q} \vee \sim\mathtt{s} \vee \mathtt{p}, \sout{\mathtt{p} \vee \mathtt{s}}, \sim\mathtt{r} \vee \sim\mathtt{q} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{r}, \mathtt{s}, \sim\mathtt{s} \vee \mathtt{t}, \mathtt{p} \vee \mathtt{t}, \sim\mathtt{s} \vee \mathtt{p}, \sim\mathtt{q} \vee \sim\mathtt{s}, \sim\mathtt{r}, \sim\mathtt{q} \vee \mathtt{p} \vee \sim\mathtt{r}\}$

Generating R_2 takes 22 inference steps but 7 resolvents are duplicates so R_2 consists of 15 new clauses.

$S_2 = S_1 \cup R_1$

$R_3 = \{\sout{\mathtt{p} \vee \sim\mathtt{s}}, \mathtt{p} \vee \sim\mathtt{r} \vee \mathtt{t}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \mathtt{p} \vee \sim\mathtt{r}, \sout{\mathtt{s}}, \sout{\sim\mathtt{s} \vee \mathtt{t}}, \sout{\sim\mathtt{q} \vee \sim\mathtt{s}}, \sout{\sim\mathtt{r}}, \mathtt{t}, \sim\mathtt{s}, \sout{\sim\mathtt{q} \vee \sim\mathtt{r}}, \sout{\sim\mathtt{q} \vee \sim\mathtt{s}}, \sim\mathtt{q} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{q}, \sim\mathtt{q}, \sim\mathtt{q} \vee \mathtt{p}, \sout{\mathtt{p} \vee \sim\mathtt{r} \vee \mathtt{t}}, \sout{\mathtt{s}}, \mathtt{p} \vee \sim\mathtt{q} \vee \sim\mathtt{r}, \sout{\sim\mathtt{r} \vee \mathtt{t}}, \sout{\mathtt{p} \vee \sim\mathtt{r}}, \sim\mathtt{q} \vee \sim\mathtt{r}, \sout{\mathtt{p} \vee \sim\mathtt{r} \vee \mathtt{t}}, \sout{\sim\mathtt{s} \vee \mathtt{t}}, \sout{\sim\mathtt{r} \vee \mathtt{t}}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \sout{\mathtt{p} \vee \sim\mathtt{q} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \mathtt{p}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \sout{\sim\mathtt{r} \vee \mathtt{t}}, \sout{\sim\mathtt{s}}, \sout{\mathtt{p} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \mathtt{p} \vee \sim\mathtt{q} \vee \mathtt{t}, \sout{\mathtt{p}}, \sout{\mathtt{p}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \sout{\sim\mathtt{q} \vee \sim\mathtt{r}}, \sout{\sim\mathtt{s}}, \sout{\mathtt{p}}, \sout{\mathtt{p} \vee \sim\mathtt{q} \vee \mathtt{t}}, \sout{\sim\mathtt{q} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\sim\mathtt{q} \vee \mathtt{t}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\sim\mathtt{q}}, \sout{\mathtt{p} \vee \sim\mathtt{q} \vee \mathtt{t}}, \sout{\sim\mathtt{q} \vee \mathtt{t}}, \sout{\mathtt{t}}, \sout{\mathtt{p} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{s}}, \sout{\mathtt{p} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \mathtt{t}}, \sout{\mathtt{p} \vee \sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\mathtt{p} \vee \mathtt{t}}, \sout{\mathtt{p}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\mathtt{p} \vee \mathtt{t}}, \sout{\sim\mathtt{s}}, \sout{\sim\mathtt{q} \vee \mathtt{t}}, \sout{\mathtt{p}}, \square, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\sim\mathtt{r}}, \sout{\mathtt{p} \vee \sim\mathtt{q}}, \sout{\mathtt{t}}, \sout{\mathtt{p}}, \sout{\sim\mathtt{q}}\}$

Generating R_3 takes 71 inference steps but 59 resolvents are duplicates so R_3 consists of 11 new clauses one of which is the empty clause ($\square$) and so a refutation has been found.

In total, 100 inference operations were required of which 31 produced new resolvents. The proof consists of only 6 inference operations as shown in Figure 2.1.

[1] Any duplicate clause or tautology generated is shown with a strike through to indicate that it is not kept

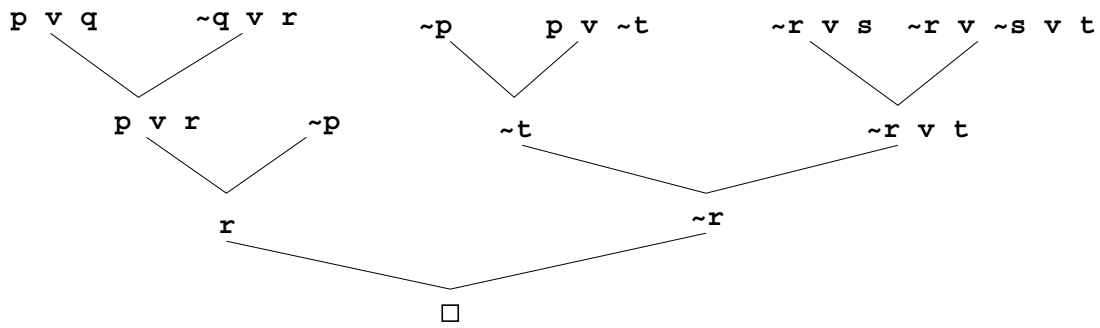

Figure 2.1: Resolution proof for Example 2

Linear resolution was developed independently by Loveland [24] and Luckham [27]. The motivation behind this approach was to restrict the number of inferences made at each level of the proof process, and thereby reduce the overall number of inferences needed to find the proof. In linear resolution, a clause is chosen from the input clause set to be the top centre clause. The top clause is resolved with an input clause and the result is a new centre clause. Resolutions can only take place between the current centre clause and a clause from the input clause set or a centre clause that occurs before the current centre clause in the deduction. (The non-centre clause used in the resolution will be referred to as a side clause.) Resolution with a previous centre clause is called an ancestor resolution. Linear resolution reduces the size of the search space dramatically at each level but often results in much deeper searches. The main advantage of this method is that it is strongly backward chaining, provided that the top centre clause is chosen appropriately, i.e., a clause generated from the negated theorem.

Figure 2.2 shows the search tree for a linear resolution deduction to depth 6 for clause set S (shown in Example 2) when $\mathtt{p} \vee \mathtt{q}$ is chosen to be the top clause. The search tree shows all possible paths taken in the deduction, with each branch representing a possible linear deduction. The nodes on a branch are the centre clauses that are created. The branch segments are labeled with the side clauses used in the resolutions with a node to produce the next node. If the side clause is an ancestor clause, it is enclosed in a box.

The full tree is not shown as it is too large. When a branch has been truncated, the number of nodes that are in the truncated section is noted at the leaf. All branches were taken to the same depth. In this example, a refutation is found at depth 6. In total, a possible 186

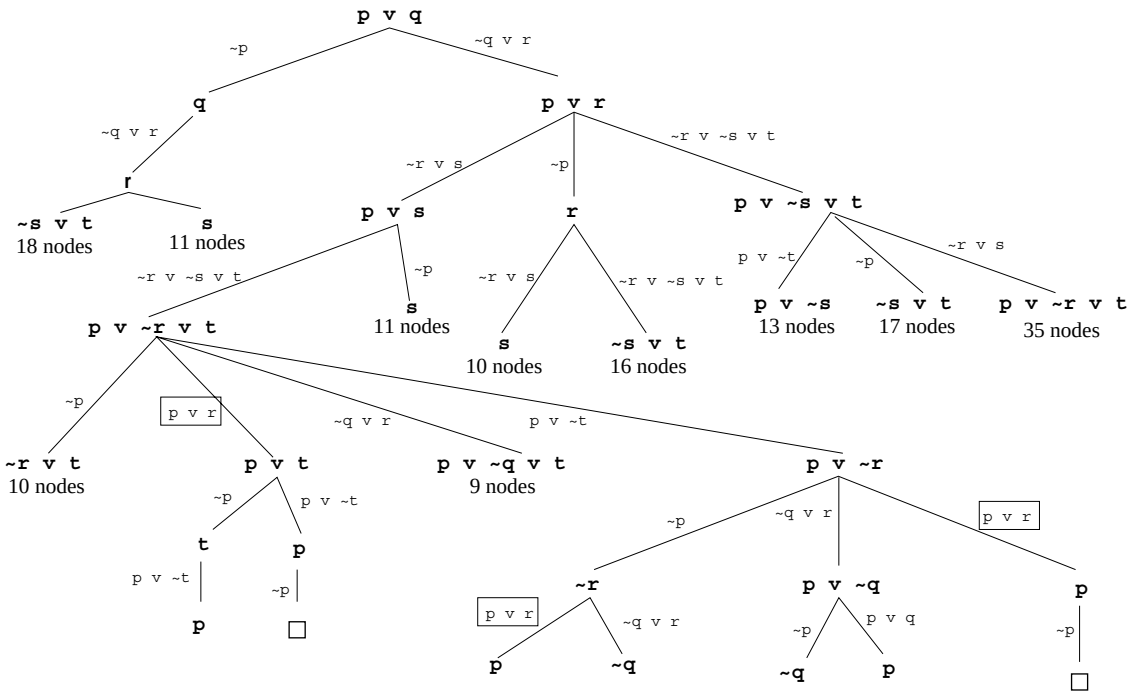

Figure 2.2: Search tree for linear resolution to depth 6

inferences are needed to compute the search space to a depth of 6.

SL-resolution [20] is linear resolution with a selection function. The selection function chooses the order in which literals can be resolved. This reduces the number of possible inference steps at each level thus further decreasing the size of the search space. It has a further restriction that says if an ancestor literal exists that is complementary to a current centre clause literal then an ancestor resolution must be performed. For example, for the same set S and top clause $\mathtt{p} \vee \mathtt{q}$, if the selection function used imposes a lexical ordering on the literals in the clauses and only allows resolutions on the rightmost literal in the centre clause then the search tree shown in Figure 2.2 reduces to that shown in Figure 2.3. At the first level, in linear resolution two inferences are possible: $\mathtt{p} \vee \mathtt{q}$ can be resolved with $\sim\mathtt{p}$ to give $\mathtt{q}$ (see the left branch in Figure 2.2) or it can be resolved with $\sim\mathtt{q} \vee \mathtt{r}$ to give $\mathtt{p} \vee \mathtt{r}$ (see the right branch in Figure 2.2). In SL-resolution, the selection function restricts the possible inference steps to those involving the rightmost literal in the centre clause, $\mathtt{q}$, which makes the resolution with $\sim\mathtt{q} \vee \mathtt{r}$ the only possible step (see Figure 2.3). Notice in Figure 2.3 at level 3, the centre clause $\mathtt{p} \vee \sim\mathtt{r} \vee \mathtt{t}$ is involved in an ancestor

resolution on r and not on t which is the literal selected by the selection function. This is because of the "compulsory" ancestor resolution rule. The search tree for SL-resolution contains 11 inferences compared to 186 for plain linear resolution.

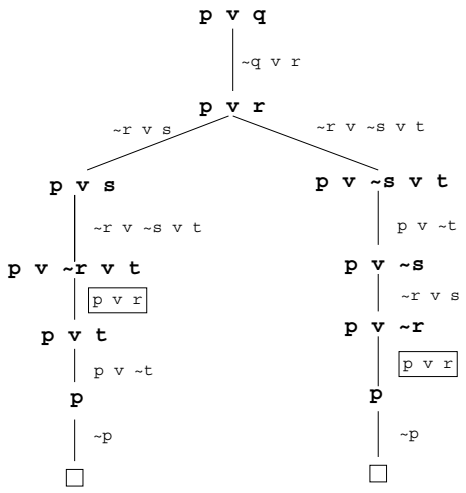

Figure 2.3: Search tree for SL-resolution to depth 6

Linear input resolution [7] was developed in 1970. Linear input resolution is a variation on linear resolution where ancestor resolutions are not allowed. All resolutions take place between a centre clause and a clause from the input clause set. This extra restriction makes linear input resolution very efficient but at a cost. Linear input resolution is complete only for Horn clause sets. Figure 2.4 shows the search tree to depth 7 for a linear input resolution deduction for the clause set S. Depth 7 contains the first occurrence of $\square$.

Model Elimination (ME) [22] was first proposed in 1968, and a second paper detailing a simplified format for the procedure [23] was published in 1969. Its relationship to linear resolution was noted by Kowalski and Kuehner [20] in their paper on SL-resolution. ME uses a chain format - the disjunctions between literals are implied and are omitted. Resolutions take place between the centre chain's rightmost B-literal and chains from the input set, called *side chains*. A resolution between a centre chain and a side chain is called an *extension*. Literals in a centre chain that have been resolved are retained, and called A-literals. A-literals are indicated by a box surrounding the literal. All other literals are called B-literals. For example, the clause $\mathtt{p} \vee \mathtt{q}$ is represent by the chain pq. For the centre chain, p$\boxed{\mathtt{q}}\sim$p, p and $\sim$p are B-literals and q is an A-literal. Side chains are

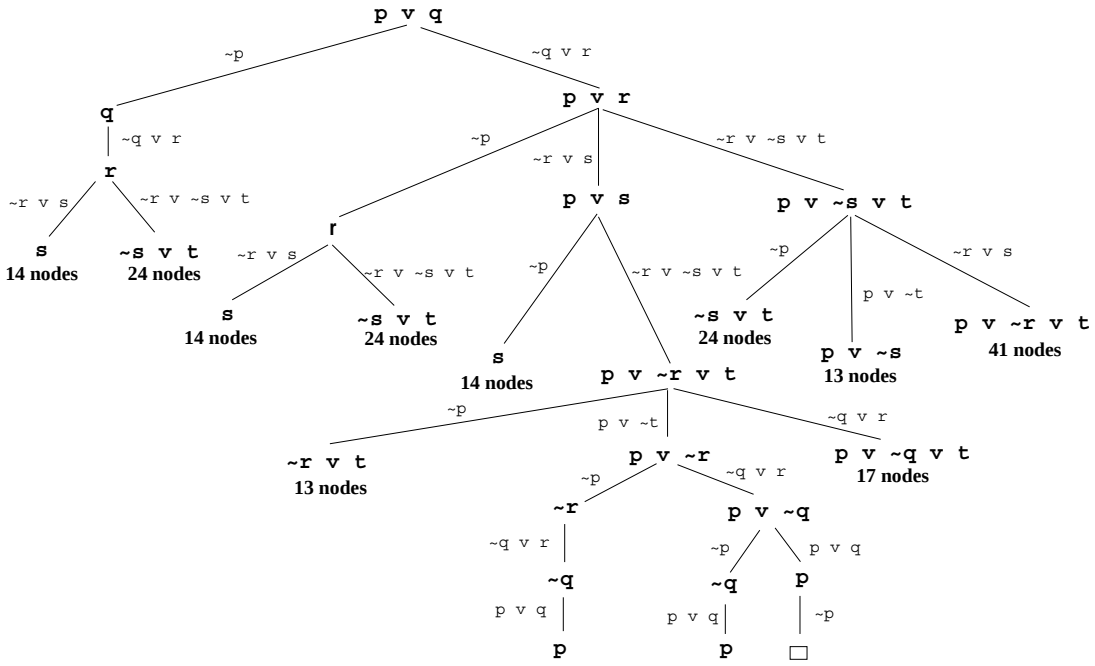

Figure 2.4: Search tree for Linear input resolution to depth 7

comprised solely of B-literals.

A-literals are used to perform ancestor resolutions without the need to carry around previously generated centre chains. When a B-literal exists to the right of a complementary A-literal in a centre chain, it can be resolved with the A-literal in an inference step called a *reduction*. The reduction step takes the place of ancestor resolutions and any associated factoring. Figure 2.5 show an ME deduction of the proof for the clause set S (using the traditional vertical format) and Figure 2.6 shows the search tree for this deduction. In ME, the literals at the right-hand end of the centre chain are operated on first. If an A-literal is at the right-hand end then it is removed in a *truncation* step.

2.2 Prolog Technology Theorem Provers (PTTP)

Prolog, a programming language based on Horn clause logic [19], was developed in 1975. Its inference engine is linear-input resolution. Most implementations use an unsound unification algorithm (no occurs check), for reasons of efficiency, and an unbounded depth-

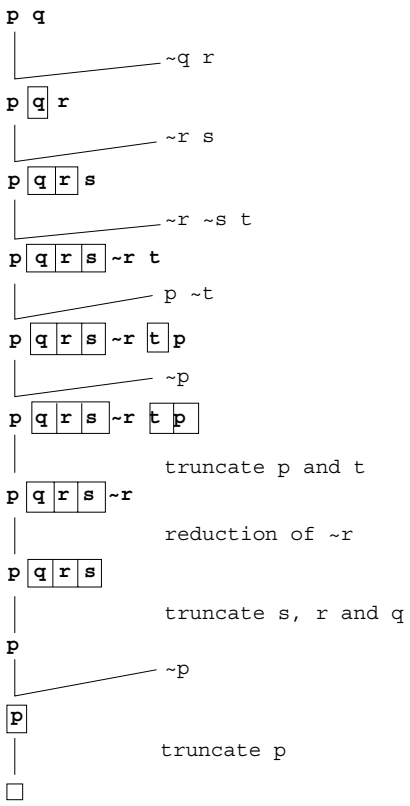

Figure 2.5: Model Elimination deduction for $S = \{\mathtt{p} \vee \mathtt{q}, \sim\mathtt{p}, \sim\mathtt{q} \vee \mathtt{r}, \sim\mathtt{r} \vee \mathtt{s}, \mathtt{p} \vee \sim\mathtt{t}, \sim\mathtt{r} \vee \sim\mathtt{s} \vee \mathtt{t}\}$

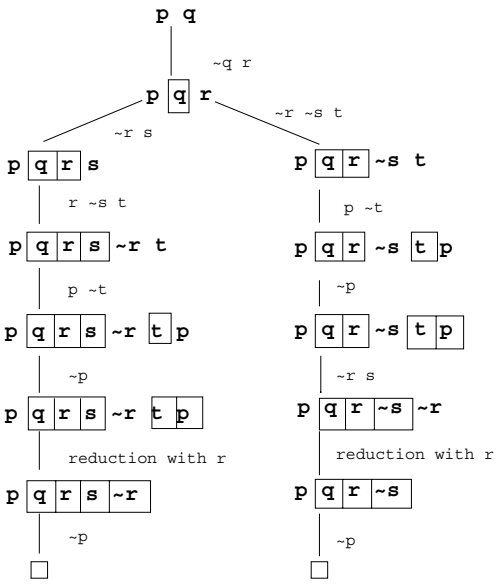

Figure 2.6: Search tree for Model Elimination to depth 6

first search strategy, which is an incomplete strategy. However, Prolog is fast and efficient, and in the early 1980's there were several papers published with a view to using Prolog or Prolog-like engines as theorem provers. Stickel proposed a Prolog technology theorem prover (PTTP) [40] as a way of extending Prolog to full linear resolution. PTTP uses ME as its basis and provides a sound unification algorithm along with a complete search strategy. In order to represent non-Horn clauses in the Prolog syntax, contrapositive clauses are generated. For example, the clause $\mathtt{p} \vee \sim\mathtt{q}$ is represented in Prolog syntax as $\mathtt{p} :\!- \mathtt{q}$. For a non-Horn clause, $\mathtt{p} \vee \mathtt{q} \vee \sim\mathtt{s}$, this would be $\mathtt{p}, \mathtt{q} :\!- \mathtt{s}$. As Prolog syntax only allows one term on the left hand side (or *head*), contrapositives are needed. The clause $\mathtt{p} \vee \mathtt{q} \vee \sim\mathtt{s}$ generates the following contrapositives:

$$\mathtt{p} :\!- \sim\mathtt{q}, \mathtt{s}$$
$$\mathtt{q} :\!- \sim\mathtt{p}, \mathtt{s}$$
$$\sim\mathtt{s} :\!- \sim\mathtt{p}, \sim\mathtt{q}$$

For a clause with n literals, n contrapositive clauses need to be generated.

The addition of contrapositives can significantly increase the input clause set and is seen as one of the main disadvanatages of the PTTP approach to theorem proving. The other disadvantage is the need to maintain a list of the A-literals for the reduction operation. This list can become quite large and add significant overhead to the system as it needs to be searched continually. Example 3 shows the contrapositives needed for the clause set S and their transformation into Prolog procedures for a PTTP deduction.

Example 3 *For* $S = \{\mathtt{p} \vee \mathtt{q}, \sim\mathtt{p}, \sim\mathtt{q} \vee \mathtt{r}, \sim\mathtt{r} \vee \mathtt{s}, \sim\mathtt{r} \vee \sim\mathtt{s} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{t}\}$ *the set* S_C *of contrapositives is as follows:*

$$\mathtt{p} :\!- \sim\mathtt{q}$$
$$\mathtt{q} :\!- \sim\mathtt{p}$$
$$\sim\mathtt{p}$$
$$\mathtt{r} :\!- \mathtt{q}$$
$$\sim\mathtt{q} :\!- \sim\mathtt{r}$$

$s :- r$

$\sim r :- \sim s$

$t :- r, s$

$\sim r :- s, \sim t$

$\sim s :- r, \sim t$

$p :- t$

$\sim t :- \sim p$

The Prolog procedures are as follows:

```
p(Ancestors) :- not_q([not_p|Ancestors]).
q(Ancestors) :- not_p([not_q|Ancestors]).
not_p(Ancestors).
r(Ancestors) :- q([not_r|Ancestors]).
not_q(Ancestors) :- not_r([q|Ancestors]).
s(Ancestors) :- r([not_s|Ancestors]).
not_r(Ancestors) :- not_s([r|Ancestors]).
t(Ancestors) :- r([not_t|Ancestors]),
                s([not_t|Ancestors]).
not_r(Ancestors) :- s([r|Ancestors]),
                    not_t([r|Ancestors]).
not_s(Ancestors) :- r([s|Ancestors]),
                    not_t([s|Ancestors]).
p(Ancestors) :- t([not_p|Ancestors]).
not_t(Ancestors) :- not_p([t|Ancestors]).
```

Prolog clauses to perform the reduction operation are also needed:

```
p(Ancestors) :- member(p,Ancestors).
q(Ancestors) :- member(q,Ancestors).
r(Ancestors) :- member(r,Ancestors).
s(Ancestors) :- member(s,Ancestors).
t(Ancestors) :- member(t,Ancestors).
not_p(Ancestors) :- member(not_p,Ancestors).
not_q(Ancestors) :- member(not_q,Ancestors).
not_r(Ancestors) :- member(not_r,Ancestors).
not_s(Ancestors) :- member(not_s,Ancestors).
not_t(Ancestors) :- member(not_t,Ancestors).
```

The search strategy used in PTTP is depth-first iterative deepening. PTTP uses a depth-first search to a depth bound which is increased by some increment if a proof is not found, and the search is then rerun with the new depth bound. Although this means recomputing results from previous levels with each depth bound increase, because of the exponential growth in the size of the search space it has been shown that for depth-first iterative deepening search "in general, it is still only a constant factor more expensive than breadth-first search" [41].

Stickel's work to extend Prolog to full first order logic lead to other systems such as Near-Horn Prolog [25] and Non-Horn Prolog [32]. One of the main problems with a PTTP style prover is the generation of contrapositives. As well as increasing the number of clauses, this can lead to 'unnatural' search behaviour. To illustrate this, consider the example originally given in [32] of the clause $and(X,Y) \leftarrow is\_true(X), is\_true(Y)$. In a PTTP style prover, the following contrapositive would be generated:

$$\texttt{not\_is\_true(X)} :- \sim \texttt{and(X,Y)}, \texttt{is\_true(Y)}.$$

Looking at this new clause in isolation, it appears to say that in order to prove that $\texttt{not\_is\_true(X)}$ is true, it is required to first prove that $\texttt{not\_and(X,Y)}$ is true for an unrelated Y and then to prove that $\texttt{is\_true(Y)}$ is true. This does seem like an odd approach to take. Both Near-Horn Prolog and Non-Horn Prolog use case-analysis as the means

of handling multi-literal heads without the need to generate all contrapositive clauses as done in PTTP and thus avoid this undesirable behaviour.

Case analysis works as follows: the clause $\mathtt{p} \vee \mathtt{q}$ can be interpreted as "$\mathtt{p}$ may be false as long as $\mathtt{q}$ is true ".

Given $S = \{\mathtt{p} \vee \mathtt{q}, \sim\mathtt{p}, \sim\mathtt{q} \vee \mathtt{r}, \sim\mathtt{r} \vee \mathtt{s}, \sim\mathtt{r} \vee \sim\mathtt{s} \vee \mathtt{t}, \mathtt{p} \vee \sim\mathtt{t}\}$

- Assume $\sim\mathtt{p}$ is true, then $\mathtt{p} \vee \mathtt{q}$ is true only if $\mathtt{q}$ is true.
  At this point, instead of following the reasoning and examining the clause $\sim\mathtt{q} \vee \mathtt{r}$, the deduction is stopped and restarted with the original conjecture. The truth of the second head literal $\mathtt{q}$ is not contested.
- Given that $\sim\mathtt{p}$ is true, $\mathtt{p} \vee \sim\mathtt{t}$ is true only if $\sim\mathtt{t}$ is true.
- If $\sim\mathtt{t}$ is true then $\sim\mathtt{s}$ is true or $\sim\mathtt{r}$ is true.
- If $\sim\mathtt{s}$ is true, $\sim\mathtt{r}$ is true.
- If $\sim\mathtt{r}$ is true, then $\sim\mathtt{q}$ is true. This contradicts the condition that $\mathtt{q}$ is true for the first stage of the deduction. So the assumption that $\sim\mathtt{p}$ is true is shown to be false.

Both near-Horn Prolog and Non-Horn Prolog required changes to be made to the Prolog engine. Baumgartner and Furbach proposed a modification to ME-PTTP called restart ME [1]. This implements case-analysis to avoid the contrapositives in a manner compatible with ME and is easily added to a PTTP based prover as demonstrated by the theorem prover PROTEIN [2]. No changes are required to the underlying prolog engine.

2.3 Guidance Strategies Employed by Linear Deduction Systems

All ATP systems need some form of guidance strategy. The search space is too large to be traversed fully. Guidance strategies can be based on semantics or syntactic considerations. Historically, syntax based strategies have been more popular.

In linear systems, guidance strategies include giving preference to short side clauses as the resulting new centre clause will be smaller and so hopefully closer to the empty clause than that generated by resolution with a long side clause. An argument to justify this heuristic is that statistically there are fewer models supporting a short clause then a long clause, and that since there are many more long clauses than short ones and the eventual proof consists of clauses shorter than some (unknown) bound, it makes sense to postpone the exploration of long clauses as much as possible. An example of this is the unit preference strategy [45]. Using this strategy, if a choice exist between using a side clause of length 1 and a side clause of length > 1 in a resolution set, then the shorter clause is tried first. It is a preference rather than a rule because unit resolution (where one of the parent clauses must be a unit clause for all resolutions) is equivalent to input resolution [7] and is complete for Horn clause sets only.

Identical Ancestor pruning is another syntactic pruning strategy that can be applied to ME proofs. Under identical ancestor pruning, a deduction cannot contain identical A-literals in the same central chain. This effectively prevents solving the same sub-goal more than once in any branch of the proof. When ME is viewed as a clausal tableau, this type of pruning is called the *regularity* condition. A clausal tableau is *regular* if no literal occurs on any one branch more than once [21]. Figure 2.7 shows two ME linear deductions, one with identical ancestor pruning and the other without, and Figure 2.8 shows the same two deductions in tableau format. Identical ancestor pruning (or regularity) is a very powerful pruning technique for ME based provers.

2.4 Summary

Modern ATP techniques are mostly based on the resolution procedure. Many refinements of resolutions have been developed with a view to reducing the size of the search space generated by the level-saturation resolution approach. One of these refinements is linear resolution. The most “popular” form of linear resolution used in modern ATP systems is Model Elimination. The Model Elimination paradigm has been successfully adapted to Prolog giving rise to Prolog Technology Theorem Provers.

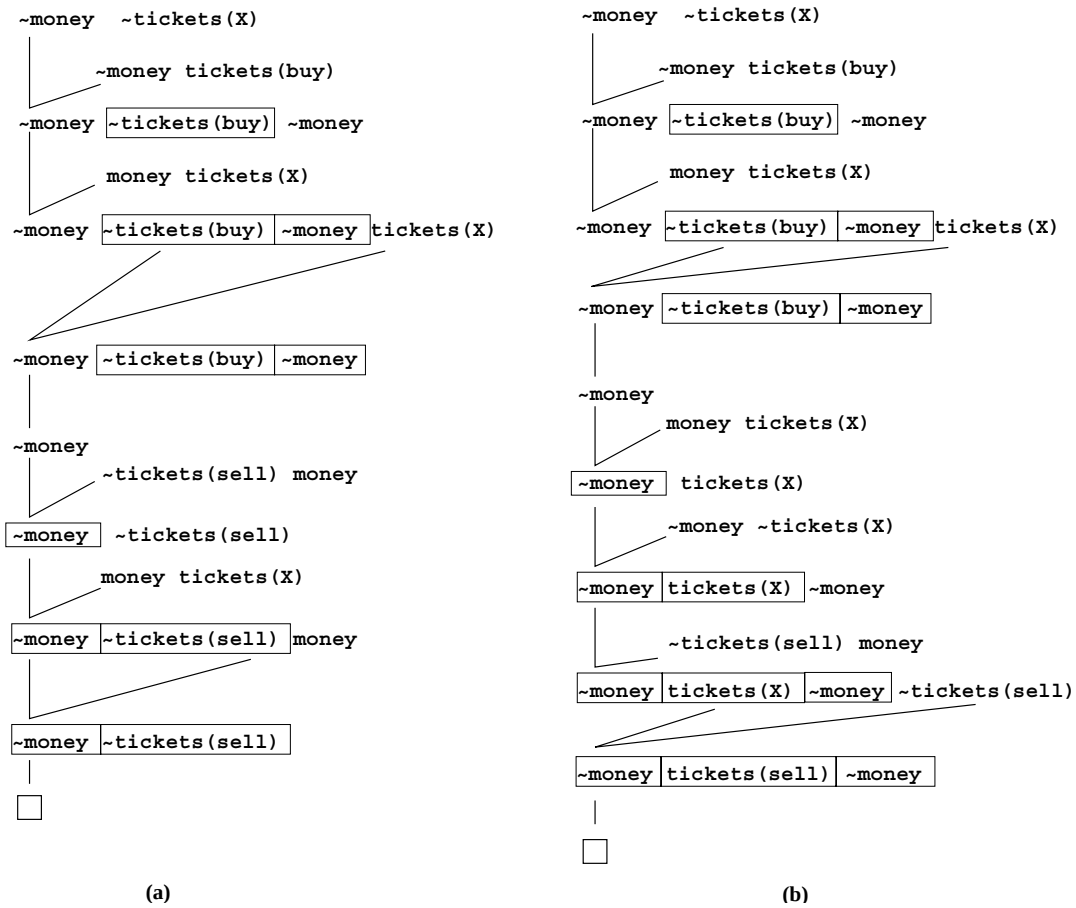

Figure 2.7: Two ME linear deductions for the clause set $\{\texttt{money} \vee \texttt{tickets(X)}, \sim\texttt{money} \vee \texttt{tickets(buy)}, \texttt{money} \vee \sim\texttt{tickets(sell)}, \sim\texttt{money} \vee \sim\texttt{tickets(X)}\}$: a deduction using identical ancestor pruning is shown in (a) and a deduction without identical ancestor pruning is shown in (b).

All ATP systems require some form of guidance strategy. Linear systems are no exception and techniques such as identical ancestor pruning, linear-input resolution and ordering strategies have been developed for these systems.

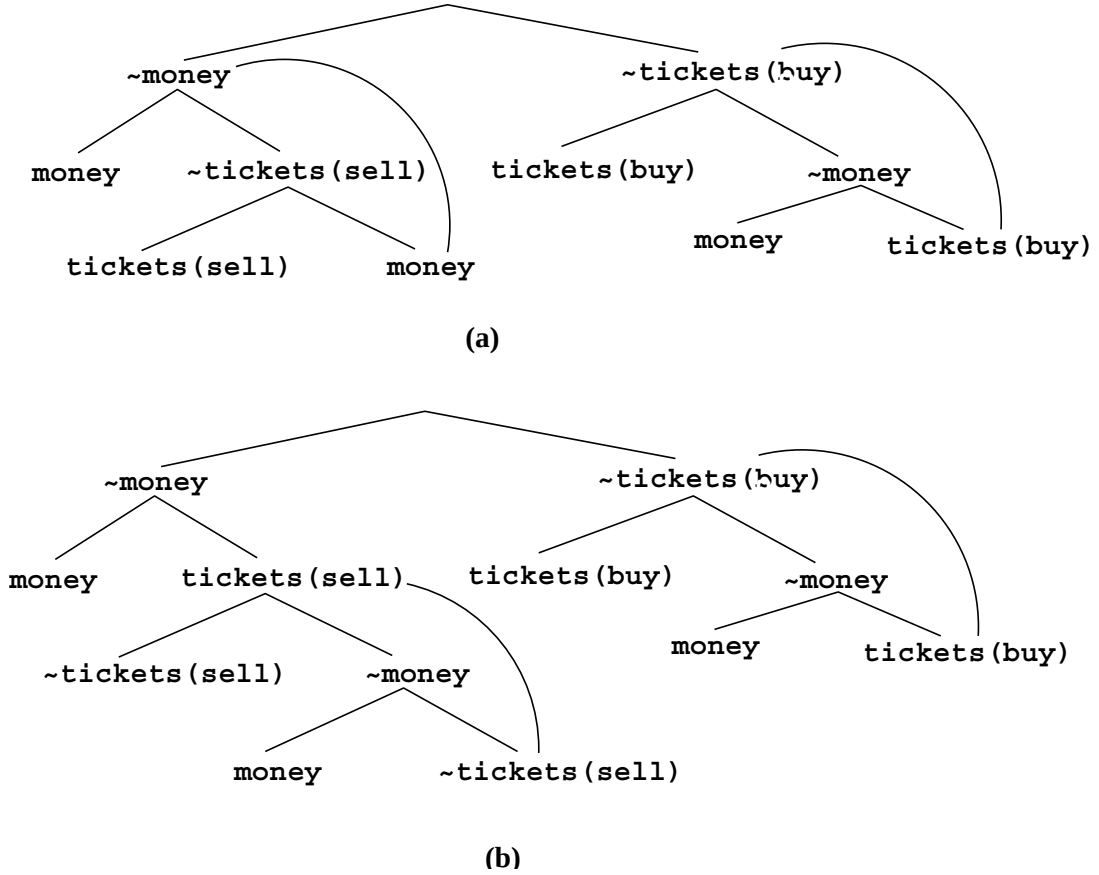

Figure 2.8: Two ME tableau deductions for the clause set $\{\texttt{money} \vee \texttt{tickets(X)},\ \sim\texttt{money} \vee \texttt{tickets(buy)},\ \texttt{money} \vee \sim\texttt{tickets(sell)},\ \sim\texttt{money} \vee \sim\texttt{tickets(X)}\}$: a regular tableau is shown in (a) and a tableau without regularity is shown in (b)

Chapter 3

Semantic Guidance Strategies

This chapter examines the history of semantic guidance from the early resolution principle based systems to the theorem proving systems of today. It also includes a brief discussion of model generation systems.

3.1 Early Resolution Systems

One of the first guidance strategies developed for the resolution procedure, the Set of Support (SoS) [45] strategy, uses semantics to introduce backward chaining reasoning. It has proven to be very successful and is still used by some of the powerful theorem provers today (e.g., OTTER [28]). The input clause set S is split into two sub sets, T and $S \backslash T$, where $S \backslash T$ is satisfiable. T is called the set of support. The restriction is then placed on resolutions such that one parent must belong to T or have an ancestor from T. It is considered to be semantic based as the set of support is usually chosen from the conjecture and hypothesis clauses and $S \backslash T$ contains the axioms. Logically, $S \backslash T$ contains information known/assumed to be TRUE and T contains the things we are trying to prove and (in the case of the conjecture) assumed to be FALSE. If both the hypothesis and the conjecture are included in the set of support, then the prover can reason forward from the hypothesis and backwards from the conjecture. If the set of support is restricted to the conjecture only then the system becomes purely backward chaining. It achieves good

pruning of the search space at the first level, but the effect diminishes as the search depth increases.

Chronologically, many early semantic based strategies were developed from syntax-based ones. For example, in P_1-deduction [34] the clauses are separated into positive and negative groups by examining the signs of their literals. Clauses containing only unnegated literals are termed positive and the rest negative. In P_1-deductions, only resolutions which have a parent clause from the positive group are allowed. P_p-deduction [30] extends this idea. It partitions all the atoms in the clause set into two groups, p_1 and p_2. If a clause contains only negated p_1 atoms and unnegated p_2 atoms then it is considered to be a *p*-clause. A P_p-deduction consists of resolutions where one parent is a *p*-clause. In [30] no recommendations are made as to how to select the atoms for p_1 and p_2.

Model resolution [26] was proposed in 1968. In this method, two models M_1 and M_2 are used as the basis for partitioning the clauses. Together, M_1 and M_2 contain all the literals in the Herbrand universe of the set of clauses i.e. $M_1 \bigcup M_2 = H(S)$, and M_1 and M_2 are disjoint i.e. $M_1 \bigcap M_2 = \emptyset$. The clause set S is partitioned into two sets S_1 and S_2. The set S_1 contains a clause $C \in S$ if it is satisfied by M_1 and $H(C) \bigcap M_2 = \emptyset$. The set S_2 contains those clauses satisfied by M_2. Deductions are restricted to those that have one parent from set S_1 and one parent from S_2. Resolvents are also partitioned using the models M_1 and M_2. One can see that if M_1 is the set of all positive literals in $H(S)$ and M_2 contains all the negative literals then the partitioning of the clauses is the same as that achieved in a P_1 deduction. With model resolution, semantics can be used to divide the clauses into the two partitions instead of using syntactic features such as the sign of the literals in the clauses.

Semantic resolution [36] is a semantic based equivalent of the earlier syntactic based strategy of hyper-resolution [34]. In hyper-resolution a negative clause is chosen as one parent and called the nucleus. Positive clauses, called electrons, are chosen to resolve against the negative literals in the nucleus. A single hyper-resolution step involving a nucleus containing n negative literals may involve up to n electrons. The result of a hyper-resolution step is a positive clause or the empty clause. In semantic resolutions, instead of the electrons being positive clauses, they are clauses that are (sometimes) FALSE in some

chosen model M. The word "sometimes" is needed as the clauses are in first order logic and so contain variables. For an electron, while one instantiation of the clause may be FALSE in the model M, another may not. Variables in clauses are (implicitly) universally quantified, so as long as it is **possible** for the electron to be FALSE, it is **to be** FALSE.

3.2 Linear Systems

With the development of linear resolution, different ways of using semantic information for search guidance were investigated. Linear input resolution [7], with its lack of complicating ancestor resolutions, lends itself to semantic pruning [5] in the following way: The empty clause, $\square$, has the interpretation of FALSE in every interpretation. A FALSE resolvent must have one or more FALSE parents. Assume there is a refutation, R, of a set of clauses, S. Assume there is a model M of the side clauses in R. The last centre clause in R is $\square$, which is FALSE in M. The side clause parent of $\square$ is TRUE in M, so the second last centre clause, C_n, must be FALSE. If C_n is FALSE then its predecessor, C_{n-1}, must also be FALSE - and so on. From this the initial top centre clause,C_1, is shown to be FALSE. So if the side clauses are known, a model of them, M, can be found and any centre clause that is TRUE in M can be rejected. For the Horn case, if we take a negative clause to be the initial centre clause, then the side clauses are the non-negative clauses and a model can be found for these.

Linear input resolution is complete for Horn clause sets only. Extending the semantic pruning idea to non-Horn clauses has been discussed. "It may be possible to adapt semantic checking to non-Horn clauses. For instance, by insisting that if a true clause is introduced attempts are made to ancestor resolve it or its descendants with a non-true ancestor. To the best of my knowledge no one has attempted this." [Alan Bundy [5], p. 149].

One way of extending semantic pruning to the non-Horn case is by using linear-input subset analysis [43]. In many linear deductions, there are situations where ancestor resolutions do not take place. These are known as linear-input subdeductions. By identifying

these situations, it is possible to use semantic pruning on these sections in the same manner as in linear-input resolution.

3.3 Modern ATP systems

Semantic guidance has been used in forward chaining systems for sometime. In semantic resolution a model is used to partition the clauses into two distinct sets. This model has to be provided to the theorem prover. "One might hope that someday the program could devise its own models." [36] SCOTT [38] and RAMCS [6] both attempt to fulfil this wish. SCOTT initially uses SoS to partition the input clause set into the support set, U, and the remaining clauses, T, and then uses a model generator to generate a model, M, for T. As new clauses are generated, they are tested against M. If a new clause g is satisfied by M, then it is added to T. If the clause g is not satisfied by M, then SCOTT tries to find a new model N for $T \cup g$. If N exists then it becomes the new model for T and g is added to T, otherwise the model M is retained and g is added to U. Searching for models is expensive so, after a number of clauses have been evaluated (this is a configurable parameter), the model is fixed and no further improvement on the model is attempted. From this point the model is used to decide if new clauses are to be included in T or U. In this manner, SCOTT devises its own models to guide its search.

Since the first SCOTT system was developed, it has undergone a series of evolutionary changes [39, 18]. The latest system is SCOTT-5 [17]. A problem with the early SCOTT systems is that the model generated to guide the search is dependent upon the order in which new clauses are presented for inclusion in the model. In SCOTT-5, multiple models are used to overcome this problem. The guidance used in SCOTT-5 is based upon the idea that if the maximally consistent subset of clauses (MCS) derived at some point in the proof search was known then, if a clause c could be found that was inconsistent with MCS then any refutation of $MCS \cup c$ would include c. Finding sets that are maximally consistent is not feasible but near-maximally consistent sets (NMCS) can be found. A set of clauses A is said to be a near-maximally consistent set if it is consistent and there is no proper extension of A that is known to be consistent [18]. Consistency is determined by

the ability of the model generation unit to produce a model for the clause set. SCOTT-5 maintains several NMCS and uses these when selecting parent clauses for inference steps.

RAMCS uses constrained clauses (c-clauses) which are [clause:constraint] couples. The constraints are used in a simultaneous search for models and refutations. When clauses are resolved together the resulting resolvent inherits the constraints from both parent plus any new ones generated by the resolution operation. The constraints contain conditions that force a clause to be evaluated to either TRUE or FALSE in an interpretation. By ensuring that in a resolution operation one parent evaluates to TRUE and the other to FALSE, semantic resolution is applied. The model is incrementally built by the constraints created as clauses are resolved together.

Although resolution based systems dominate the field of ATP, some work has been done re-examining Herbrand-based instantiation methods. With the increased computing resources and utilizing techniques from propositional theorem proving research, some of the instantiation methods are now feasible. The CLIN-S system [10] is an instantiation based prover that uses ordered semantic hyper-linking (OSHL). An initial model M is provided along with the input clauses, and this model is used to generate ground instances of the clauses. When ground clauses are created that are not satisfied by the model, the model M is modified so that it is a model for all the ground clauses. If a set of ground clauses is found that can not be satisfied by any model than a refutation has been found.

More recently, semantic guidance has been incorporated into a resolution theorem prover using clause graphs [9]. The nodes of a clause graph are the literals. Complementary, unifiable literals are connected by a link, and literals are grouped together to form a clause. So each link represents a possible resolution step in the graph. To perform a resolution step, a link is selected, the resolution of the linked clauses is performed and the link is deleted. The new resolvent is added into the graph and new links for this clause are inserted. The deletion of the resolved upon link may result in the creation of pure literals - literals that have no links to others. Clauses containing pure literals can be deleted from the graph. Semantic guidance is used in selecting which links to act upon. A number of models are generated for the axioms in the original clause set. Each link is labeled with the number of models in which the resolvent generated from that link is TRUE. The link

with the smallest label is the one selected for the next resolution step.

3.4 Model Generation

Where does the semantics used in the systems mentioned in Section 3.3 come from? An informed user of these theorem provers may know enough about the problem they are trying to solve that they can provide their own. Some of these systems either need to generate models as they proceed through their search or provide a fully automated mode where all semantics are generated by the system. Software to generate models have been around for some time. Some of the earliest ATP systems used instantiation methods based on Herbrand's theorem [8]. They generated ground clauses from the input clause set, using the clause set's Herbrand Universe as the domain elements, and then tested the ground clauses for unsatisfiability. If the domain size is finite then the methods can be used to determine satisfiability and to generate models. The best known of these methods is the Davis-Putnam-Loveland-Logemann (DPLL) algorithm [12, 11].

The DPLL algorithm is used as the basis for several propositional model generation systems such as SATO [46] and MACE [28]. It is possible to use the DPLL algorithm to find models for first-order problems by converting the first-order clauses into propositional clauses using a finite domain. For example, given the clause $\mathtt{p}(\mathtt{f}(\mathtt{X}),\mathtt{Y}) \vee \mathtt{q}(\mathtt{X},\mathtt{g}(\mathtt{Y}))$ and the domain $\mathcal{D} = \{0,1\}$, we first need to flatten the clause to remove the functors. The original clause is replaced with a new clause as follows:

$$(\mathtt{Z} = \mathtt{f}(\mathtt{X})) \wedge (\mathtt{W} = \mathtt{g}(\mathtt{Y})) \leftarrow \mathtt{p}(\mathtt{Z},\mathtt{Y}) \vee \mathtt{q}(\mathtt{X},\mathtt{W})$$

Of course this clause in not in CNF. To convert it to CNF two new binary predicates, `predicate_f` and `predicate_g`, are created such that $\mathtt{predicate\_f}(\mathtt{X},\mathtt{Z})$ represents $(\mathtt{Z} = \mathtt{f}(\mathtt{X}))$ and $\mathtt{predicate\_g}(\mathtt{Y},\mathtt{W})$ represents $(\mathtt{W} = \mathtt{g}(\mathtt{Y}))$. The new clause in CNF is:

$$\mathtt{p}(\mathtt{Z},\mathtt{Y}) \vee \mathtt{q}(\mathtt{X},\mathtt{W}) \vee \sim\mathtt{predicate\_f}(\mathtt{X},\mathtt{Z}) \vee \sim\mathtt{predicate\_g}(\mathtt{Y},\mathtt{W})$$

The propositional clauses are generated by substituting values from $\mathcal{D}$ for the variables in the flattened clause. The DPLL algorithm is then applied to the propositional clauses and a model generated.

The conversion of first order clauses to propositional clauses results in a larger input clause set. The deeper the nesting of the functions the bigger the expansion of the clause set. In some cases this proves to be prohibitive in that the clause set becomes too large to manage in the memory available. Other first-order model generators use constraint satisfaction based algorithms, for example, FINDER [37] and SEM [47]. These systems are designed for first-order logic and don't convert clauses to the propositional state.

Model generators have been used in finite mathematics research, and also in theorem proving to provide counter-examples. More recently, FINDER has been used in the SCOTT system to generate models to provide semantic information.

3.5 Summary

This chapter has reviewed several different semantic guidance strategies that have been developed including the Set of Support, model resolution, and semantic resolution. For linear systems, semantic guidance techniques have been developed for linear-input resolution and ways of applying these to full linear resolution have been discussed. Modern ATP systems that use semantic guidance have also been reviewed including SCOTT, RAMCS, and CLIN-S.

Model generation systems were also discussed as a means of generating semantic information for use with a theorem prover. Two main types were reviewed, namely DPLL based and constraint satisfaction based systems.

Chapter 4

Guiding Linear Deductions with Semantics

This chapter describes the GLiDeS strategy for guiding a linear deduction system using semantics. In the first section the theory is explained and completeness discussed. Next the design and architecture of the implemented PTTP+GLiDeS system is described.

4.1 Theory

The GLiDeS semantic pruning strategy is based upon the strategy that can be applied to linear-input deductions (see Section 3.2). Linear-input resolution is complete only for Horn clauses and, unfortunately, the extension of the linear-input semantic pruning strategy to linear deduction is not direct. The possibility of ancestor resolutions means that centre clauses may be TRUE in a model of the side clauses. For the non-Horn case, ancestor resolution is required for refutation-completeness.

In GLiDeS, rather than placing a constraint on entire centre clauses in a refutation, a semantic constraint is placed on selected literals of the centre clauses as follows: The input clauses other than the chosen top clause of a linear deduction are named the *model clauses*. In a completed linear refutation, all centre clause literals that have resolved against input clause literals are required to be FALSE in a model of the model clauses.

TRUE centre clause literals must be resolved against ancestor clause literals. This leads to a semantic pruning strategy for ME that, at every stage of a deduction, requires all A-literals in the deduction so far to be FALSE in a model of the model clauses. The result is that only FALSE B-literals are extended upon, and TRUE B-literals must reduce.

4.1.1 Formal Notation

The notation used is that described by Baumgartner and Furbach [3]. Before formally describing the GLiDeS pruning strategy, it is necessary to redefine Model Elimination in terms of the new notation.

New Terminology

The ME deduction in Figure 4.1 is shown in tableau format as a tree structure. Extension operations are performed by resolving a leaf node with an input clause resulting in the leaf node becoming a new internal node and the literals from the extending clause becoming new leaf nodes. The leaf node and the literal from the extending clause are said to have a *connection* if there exists a MGU σ such that when σ is applied to the leaf node and the literal from the extending clause they are then complementary. If a leaf node is complementary to one of the internal nodes in its branch then the branch contains a contradiction and is considered *closed*, indicated with an asterisk. Once a branch is closed it need not be considered further. If a tableau deduction consists of all closed branches then a refutation has been found.

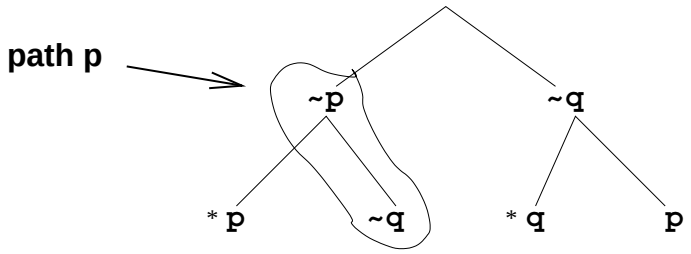

Figure 4.1: Tableau ME proof for the clause set $\{\mathtt{p} \vee \mathtt{q}, \mathtt{p} \vee \sim\mathtt{q}, \sim\mathtt{p} \vee \mathtt{q}, \sim\mathtt{p} \vee \sim\mathtt{q}\}$

A *path* is an open branch and can be described as a sequence of literals from the root of

the tableau to a leaf node. Given the ME deduction shown in Figure 4.1, the path p can be described as $p = \sim\mathtt{p} \circ \sim\mathtt{q}$ where $\circ$ is the append function for literal sequences. The last literal in a path p is called the *leaf of p* or $leaf\{p\}$. A ME tableau consists of a multiset (or bag) of paths. Definition 1 gives the formalization of the Model Elimination procedure in terms of this notation.

Definition 1 (Model Elimination) *Given a set of clauses C, a sequence $(\mathcal{P}_1, ..., \mathcal{P}_n)$ is called an ME derivation iff*

- *$\mathcal{P}_1$ is a path multiset $\{< L_1 >, ..., < L_n >\}$ consisting of paths of length 1, with $L_1 \vee ... \vee L_n$ in C, and*
- *$\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of an extension step, or*
- *$\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by means of a reduction step.*

A refutation is a derivation where $\mathcal{P}_n = \{\}$.

The extension inference rule is defined as

$$\frac{\mathcal{P} \cup \{p\} \qquad L \vee D}{\mathcal{R}}$$

where

1. *$\mathcal{P} \cup \{p\}$ is a path bag, and $L \vee D$ is a variable disjoint variant of a clause in C; L is a literal and D denotes the remaining literals of $L \vee D$.*
2. *$(leaf(p), L)$ is a connection with MGU σ.*
3. *$\mathcal{R} = (\mathcal{P} \cup \{p \quad \circ < K > \mid K \in D\})\sigma$*

The reduction inference rule is defined as

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}_\sigma}$$

where

1. $\mathcal{P} \cup \{p\}$ *is a path bag.*
2. *There is a literal* L *in* p *such that* $(L, leaf(p))$ *is a connection with MGU* σ.

The GLiDeS strategy can be incorporated into this definition for tableaux ME.

Definition 2 (ME + GLiDeS) *Given a set of clauses* C *and an interpretation* $\mathcal{I}$*, a sequence* $(\mathcal{P}_1, ..., \mathcal{P}_n)$ *is called an GLiDeS ME derivation iff*

- $\mathcal{P}_1$ *is a path multiset* $\{< L_1 >, ..., < L_n >\}$ *consisting of paths of length 1, with* $L_1 \vee ... \vee L_n$ *in* C *and* $L_1 \vee ... \vee L_n$ *is FALSE in* $\mathcal{I}$ *, and*
- $\mathcal{P}_{i+1}$ *is obtained from* $\mathcal{P}_i$ *by means of an extension step, or*
- $\mathcal{P}_{i+1}$ *is obtained from* $\mathcal{P}_i$ *by means of a reduction step.*

A refutation is a derivation where $\mathcal{P}_n = \{\}$.

The extension inference rule is defined as

$$\frac{\mathcal{P} \cup \{p\} \qquad L \vee D}{\mathcal{R}}$$

where

1. $\mathcal{P} \cup \{p\}$ *is a path bag, and* $L \vee D$ *is a variable disjoint variant of a clause in* C*;* L *is a literal and* D *denotes the remaining literals of* $L \vee D$.
2. $(leaf(p), L)$ *is a connection with MGU* σ.
3. $leaf(p)$ *has a FALSE ground instance in* $\mathcal{I}$.
4. $\mathcal{R} = (\mathcal{P} \cup \{p \quad \circ < K > \mid K \in D\})\sigma$

The reduction inference rule is defined as

$$\frac{\mathcal{P} \cup \{p\}}{\mathcal{P}_\sigma}$$

where

1. *$\mathcal{P} \cup \{p\}$ is a path bag.*
2. *There is a literal L in p such that $(L, leaf(p))$ is a connection with MGU σ.*
3. *$L\sigma$ has a FALSE ground instance in $\mathcal{I}$.*

4.1.2 Completeness and Soundness

The GLiDeS strategy is sound as it is a search space pruning strategy and doesn't change any inference operations - it just removes from the search some branches of the search path. Completeness is a more complicated issue.

The GLiDeS strategy is complete when the guiding model produces a *semantic-Horn* set of ground clauses from the input clause set. By semantic-Horn we mean that when the input clause set is made ground by substitution of the model's domain elements and evaluated in the guiding model then there is at most one TRUE literal per clause. A measure of how close to semantic-Horn a clause set is for a particular guiding model is the *excess TRUE literal count* which is analogous to the excess positive literal count used to measure closeness to a Horn set. The excess TRUE literal count, k, is the sum of the number of "extra" TRUE literals in a clause. To be semantic Horn a clause may have at most 1 TRUE literal, any above this is excess and contributes to k.

The completeness proof for ME+GLiDeS for a clause set and guiding model with $k = 0$ is shown below and is essentially the similar to the proof by Slagle for renaming [36].

Theorem 1 *Let S be a minimally unsatisfiable set of clauses and M is a finite model of $S \backslash C$ where C is the chosen top clause that is FALSE in M. If S is semantic-Horn with respect to M then there exists a GLiDeS-deduction of S.*

Proof

Let S' be some domain ground instance of S. Let C' be the chosen top clause from S' for a ME deduction. Let M' be a model of $S' \backslash C'$ (the *model clauses*) and C' is FALSE in M'. M' is called the guiding model.

Let $k(S')$ denote the excess TRUE literal count for S' with respect to M'.

Let $k(S) = 0$, that is all clauses in S' have 0 or 1 TRUE literal in M'.

Rename the literals in S' such that all literals that are TRUE in M' are positive and all literals that are FALSE in M' are negative. Let this renaming of S' be called S'_r, the top clause C' becomes C'_r, and the corresponding guiding model, M'_r.

Then there exists a linear input refutation of S'_r [16] i.e. a ME refutation without reductions. As C'_r is a negative clause and each clause in S'_r contains at most 1 positive literal, all A-literals are negative, i.e., all A-literals are FALSE in M'_r. Therefore, there exists an ME refutation of S' without reductions and all A-literals are FALSE in M'. That is there exists a GLiDeS refutation of S' with respect to M'.

If there is a GLiDeS refutation of S' then there is a GLiDeS refutation of S as each clause $C' \in S'$ corresponds to some $C \in S$. This completes the proof. □

For $k(S) > 0$, when combined with regularity pruning the GLiDeS strategy is not complete.

Example 4 $S = \{\neg\mathtt{p} \vee \neg\mathtt{r} \vee \neg\mathtt{s}, \mathtt{p} \vee \mathtt{s} \vee \mathtt{r}, \neg\mathtt{p} \vee \mathtt{t} \vee \mathtt{u}, \mathtt{p} \vee \neg\mathtt{t} \vee \neg\mathtt{u}, \neg\mathtt{q} \vee \mathtt{r}, \neg\mathtt{q} \vee \mathtt{s}, \neg\mathtt{q} \vee \mathtt{t}, \neg\mathtt{q} \vee \mathtt{u}, \mathtt{q} \vee \neg\mathtt{r}, \mathtt{q} \vee \neg\mathtt{s}, \mathtt{q} \vee \neg\mathtt{t}, \mathtt{q} \vee \neg\mathtt{u}\}$

The top clause is $\neg\mathtt{p} \vee \neg\mathtt{r} \vee \neg\mathtt{s}$ *and remainder of clauses are the model clauses. The guiding model is* $M = \{\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}\}$

For S *in model* M, $k(S)$ *is 3.*

A GLiDeS proof with regularity doesn't exist for this set S. The problem is the clause $\mathtt{p} \vee \mathtt{s} \vee \mathtt{r}$ which contains 3 TRUE literals. For a GLiDeS proof to exist this clause needs to be at the end of a branch that enables the extra TRUE literals to reduce against FALSE internal nodes. The problem appears to be regularity pruning which is a common pruning method employed in ME deduction systems. For the problem set given in Example 4 however, a GLiDeS proof is possible if we ignore regularity (see Figure 4.3).

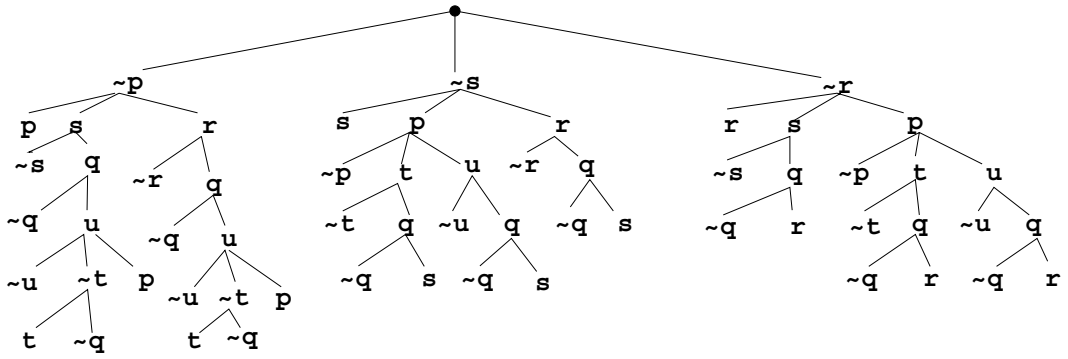

Figure 4.2: non-GLiDeS proof with regularity

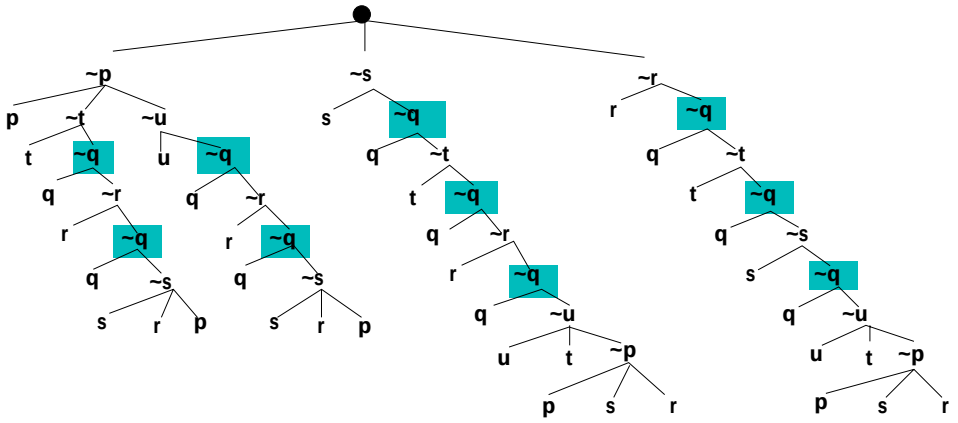

Figure 4.3: GLiDeS proof without regularity. Internal nodes that violate regularity are highlighted

A model elimination system would not be implemented without regularity pruning, so for all practical purposes GLiDeS is an incomplete strategy.

The success of GLiDeS depends on the guiding model chosen. Unlike linear-input deductions where pruning can be successfully used with ANY model of the side clauses, in GLiDeS pruning one model of the side clauses may result in a proof being found where another may not. For example, consider the problem MSC006-1 from the TPTP library [44]. Stated in English the problem is:

> "Suppose there are two relations, P and Q. P is transitive, and Q is both transitive and symmetric. Suppose further the *squareness* of P and Q: any two things are related either in the P manner or the Q manner. Prove that either P

is total or Q is total."

Converting this problem statement to CNF FOL and negating the conjecture produces the clause set $S_{MSC006-1} = \{\sim\mathtt{p}(\mathtt{X},\mathtt{Y}) \vee \sim\mathtt{p}(\mathtt{Y},\mathtt{Z}) \vee \mathtt{p}(\mathtt{X},\mathtt{Z}), \sim\mathtt{q}(\mathtt{X},\mathtt{Y}) \vee \sim\mathtt{q}(\mathtt{Y},\mathtt{Z}) \vee \mathtt{q}(\mathtt{X},\mathtt{Z}), \sim\mathtt{q}(\mathtt{X},\mathtt{Y}) \vee \mathtt{q}(\mathtt{Y},\mathtt{X}), \mathtt{p}(\mathtt{X},\mathtt{Y}) \vee \mathtt{q}(\mathtt{X},\mathtt{Y}), \sim\mathtt{p}(\mathtt{a},\mathtt{b}), \sim\mathtt{q}(\mathtt{c},\mathtt{d})\}$.

A GLiDeS proof for this clause set is shown in Figure 4.4. The chosen top clause is $\sim\mathtt{q}(\mathtt{c},\mathtt{d})$ and the set of model clauses is $S_{MSC006-1(M)} = \{\sim\mathtt{p}(\mathtt{X},\mathtt{Y}) \vee \sim\mathtt{p}(\mathtt{Y},\mathtt{Z}) \vee \mathtt{p}(\mathtt{X},\mathtt{Z}), \sim\mathtt{q}(\mathtt{X},\mathtt{Y}) \vee \sim\mathtt{q}(\mathtt{Y},\mathtt{Z}) \vee \mathtt{q}(\mathtt{X},\mathtt{Z}), \sim\mathtt{q}(\mathtt{X},\mathtt{Y}) \vee \mathtt{q}(\mathtt{Y},\mathtt{X}), \mathtt{p}(\mathtt{X},\mathtt{Y}) \vee \mathtt{q}(\mathtt{X},\mathtt{Y}), \sim\mathtt{p}(\mathtt{a},\mathtt{b})\}$. The guid-

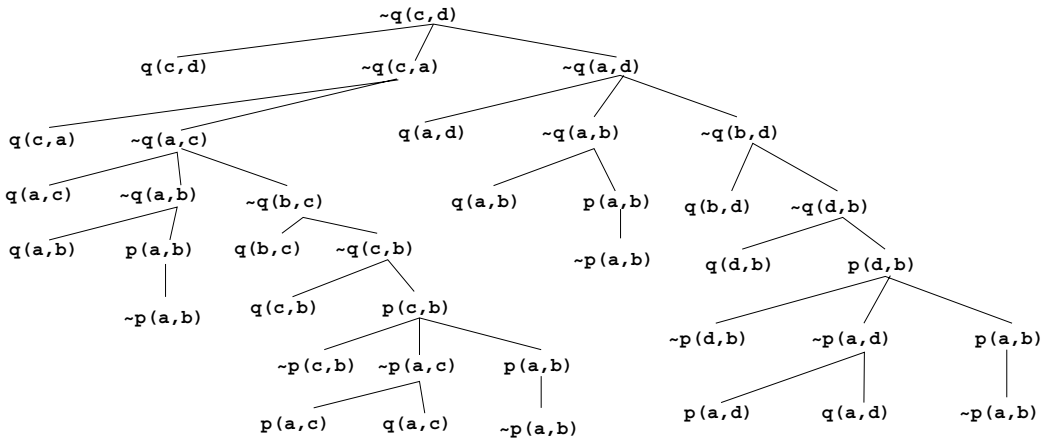

Figure 4.4: A GLiDeS deduction for MSC006-1.

ing model for this deduction is

$$M_{MSC006-1} = \left\{ \begin{array}{lll} q(X,Y) = & TRUE & \forall X,Y \in \{a,b,c,d\} \\ p(X,Y) = & \left\{ \begin{array}{ll} TRUE & \textit{for } X = a, Y \in \{d,c\} \\ FALSE & \textit{otherwise} \end{array} \right. & \end{array} \right\}$$

When generating a model for the model clauses, more than one model is possible. Alternative models generated were

$$M_{MSC006-1(2)} = \left\{ \begin{array}{lll} q(X,Y) = & TRUE & \forall X,Y \in \{a,b,c,d\} \\ p(X,Y) = & FALSE & \forall X,Y \in \{a,b,c,d\} \end{array} \right\}$$

$$M_{MSC006-1(3)} = \left\{ \begin{array}{lll} q(X,Y) = & TRUE & \forall X,Y \in \{a,b,c,d\} \\ p(X,Y) = & \left\{ \begin{array}{ll} TRUE & \textit{for } X = a, Y = d \\ FALSE & \textit{otherwise} \end{array} \right. & \end{array} \right\}$$

$$M_{MSC006\text{-}1(4)} = \left\{ \begin{array}{lll} q(X,Y) = & TRUE & \forall X,Y \in \{a,b,c,d\} \\ p(X,Y) = & \left\{ \begin{array}{ll} TRUE & \textit{for } X=a, Y=c \\ FALSE & \textit{otherwise} \end{array} \right. & \end{array} \right\}$$

Of the four guiding models, only $M_{MSC006\text{-}1}$ results in the GLiDeS guided system finding a proof. Using any of the other models, $M_{MSC006\text{-}1(2)}$, $M_{MSC006\text{-}1(3)}$ or $M_{MSC006\text{-}1(4)}$, with the GLiDeS guided system result in non-termination of the program.

Interestingly, the proof discovered using $M_{MSC006\text{-}1}$ is different from the one discovered by the unguided system (see Figure 4.5) which is a non GLiDeS proof. At first the proof shown in Figure 4.5 looks like it will be GLiDeS compliant - it contains some reductions but no complementary internal nodes. If this proof was a GLiDeS proof the guiding model

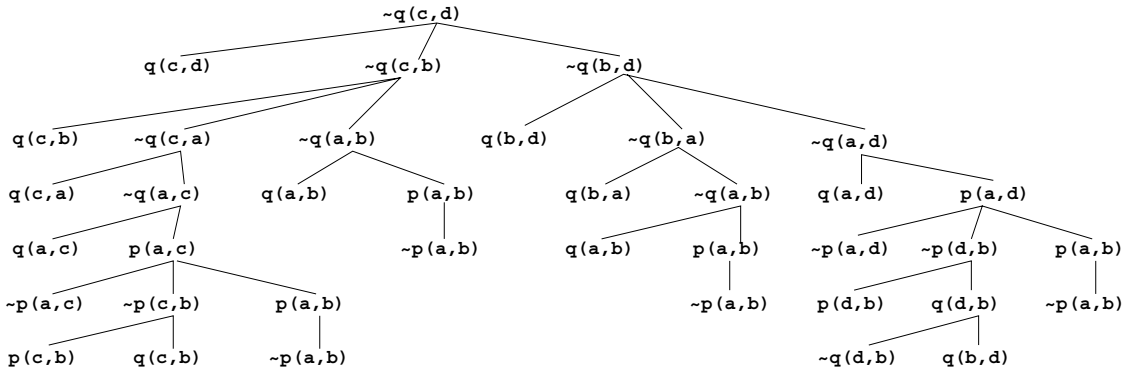

Figure 4.5: An ME deduction for MSC006-1.

would be given (at least partially) by the leaf nodes of the tableau. In this case, the leaf nodes contain the literals $\mathtt{q(b,d)}$ and $\sim\mathtt{q(d,b)}$. There is no model of the model clauses containing these two literals as the model clause $\sim\mathtt{q(X,Y)} \vee \mathtt{q(Y,X)}$ is made FALSE when $\mathtt{X} = \mathtt{b}$, $\mathtt{Y} = \mathtt{d}$ and $\mathtt{q(b,d)} = TRUE$, $\sim\mathtt{q(d,b)} = TRUE$.

MSC006-1 is a relatively uncomplicated problem. It contains no functions, only constants and variables, and it is a minimally unsatisfiable clause set - all clauses are used in producing the proof. Yet for this problem we can see that applying semantic guidance is not as straightforward as for a Horn clause set where linear input resolution can be used. There is no guarantees that any model generated will be effective. However, when an effective model is found the guidance it provides can be very effective. For example, for MSC006-1 the unguided system found a proof in 100.47 seconds and 1792624 inference steps while

the guided system found a proof in 5.25 seconds and 14308 inference steps. Given this type of improvement in performance it is worth investigation the pruning strategy further to determine:

- Strategies for selection of the "best" guiding model;
- The extent of the incompleteness problems; and
- If it is possible to identify those types of problems where the GLiDeS pruning strategy is best utilized.

4.2 GLiDeS System

The GLiDeS pruning strategy was evaluated by incorporating it into an existing linear deduction system and then combining this system with a model generator. The decision to use an existing system, rather then write a new system from scratch, was motivated by an unwillingness to "re-invent the wheel" and a desire to evaluate the effectiveness of the pruning strategy as opposed to the quality of a new theorem prover.

Figure 4.6 shows the architecture of the PTTP+GLiDeS system. PTTP+GLiDeS uses a Prolog technology theorem prover (PTTP v2e [42]) to compile the input clauses into Prolog code which is then run on a Prolog engine. An interpretation generator takes (MACE v1.4 [29]) the model clauses from the input clause set and generates a model which is also given to the Prolog engine. The Prolog code uses the model to implement the semantic guidance, as described in Section 6.1.

4.3 Summary

In this chapter, the GLiDeS strategy for semantic pruning of model elimination deductions was defined. It was shown to be sound but complete only for a subset of problems termed Semantic Horn. The GLiDeS strategy is incomplete when combined with identical

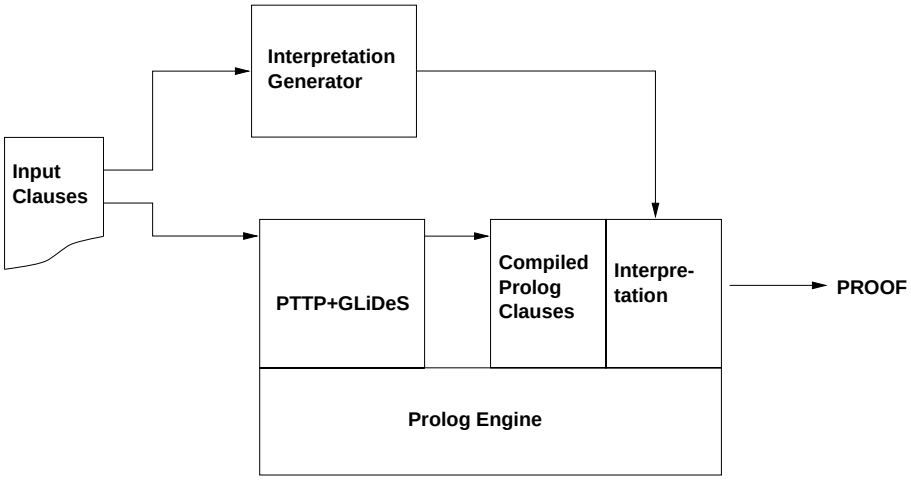

Figure 4.6: Architecture of the PTTP+GLiDeS system.

ancestor pruning (also know as regularity pruning). To evaluate the effectiveness of the GLiDeS strategy, an ATP system incorporating GLiDeS with PTTP (a Model Elimination based theorem prover) and MACE (a model generation program) was proposed. The architecture of the PTTP+GLiDeS system was described.

Chapter 5

Model Generation and Selection

One of the major problems with using semantics to guide automated theorem proving systems is the overhead involved in performing the semantic checks. However, if the semantics provide effective guidance, this is more than offset by the reduction in the search space. This leads to the question: What are good semantics and how can the *best* semantics be selected from a range of choices?

5.1 Model Generation

For a given set of model clauses, it is usually possible to generate a number of finite models. During experimentation with PTTP+GLiDeS, it was noticed that different models sometimes produced very different results. One model would yield a solution very quickly while another would fail to find a proof within the time limit. The overhead in time associated with performing the semantic checking is often large. If a model performs effective pruning of the search space this overhead is more than compensated for. If a model provides little or no guidance than better performance would be achieved by **not** using the GLiDeS strategy. The system requires a way of determining which model from a set of models, would be the best choice for using with GLiDeS.

5.2 Model Selection Heuristic

A set of clauses is said to be semantic Horn with respect to a model M if, when the clauses are instantiated with the model's domain elements and evaluated in the model, there is at most one TRUE literal in each clause. It has been shown in Section 4.1.2 that if a model of the model clauses produces a *semantic Horn set* of ground clauses for a given input clause set then PTTP+GLiDeS is refutation complete. Given a number of different models for a model clause set, it is hypothesized that the model which produces a ground clause set closest to semantic Horn will be the best choice for using with GLiDeS.

In order to determine which model produces a ground clause set closest to semantic Horn, the number of *excess true literals* for each set of ground clauses needs to be measured. The excess true literal count is similar to the excess positive literal count used to measure how near to Horn a set of clauses is. Once the clause set has been grounded using the model's domain, the ground clauses are evaluated in the model. If a clause contains 0 or 1 TRUE literal then it is considered semantically Horn. Any TRUE literals after the first are considered to be excess. For example, a ground clause with 4 TRUE literals has an excess TRUE literal count of 3. The total excess TRUE literal count for a clause set is the sum of the individual clause excess TRUE literals. This gives a measure of how far from semantically Horn a set of clauses are.

When generating models it is possible to specify the domain size to use. Some model generators (e.g. SEM, FINDER, MACE2, MACE4) will output all models found with a domain size within a given range (lower and upper bound provided as input data). Other generators (e.g. MACE) look for models of a fixed domain size. PTTP+GLiDeS uses MACE as its model generation component and so models generated for a particular clause set all have the same domain size. It is possible to run MACE with different domain sizes to build a collection of models with various domain sizes. However, problems occur when trying to compare the total excess TRUE literal counts of models with different domain sizes.

If different models for the same clause set have different domain sizes then the number

of ground clauses generated will be different and so comparing straight excess true literal counts may not be "fair". Normalizing the TRUE literal count to give an "average TRUE literal count per clause" has been considered but is not deemed to be reliable. Early experiments indicate that the models with smaller domains generally have a lower normalized TRUE literal count and so the normalized TRUE literal count is biased to selection of smaller models which are not necessarily the ones that provide the best guidance. To ensure fair comparison, all models for a particular clause set are generated with the same domain size.

Tuning of the domain size for a model can have a large impact on the types of models generated and thus on the performance of the guidance. Ideally, a user of the system would have a good "feel" for the problem (s)he is trying to solve and could intelligently choose an appropriate setting. For testing of the model selection heuristic it is desirable to examine a large number of problems and so individually setting the domain size for each problem becomes unrealistic. It was necessary to have a default setting to enable large scale testing.

Initially, the default MACE setting (starting with a domain size of two and incrementing by one until one or more models were found) was tried. For some clause sets, this approach works well but for many it did not. After thinking about what the domain of a model represents it was decided to set the domain size equal to the number of constants (including Skolem constants) in the clause set. The rationale for this is that it was felt that the constants were placed there by the problem's author and so in some way represent the domain of the problem. Of course, Skolem constants are added by the conversion from FOL to CNF and so represent something of an artefact, but at this stage they are considered equivalent to "normal" constants.

Another reason for starting the model search at the number of constants is that the more detailed a model, the more able it is to distinguish between different terms. For example, a model with a domain size of 2 for a clause set with three constants a, b, c might assign the following values, $a = 0, b = 1, c = 0$, thus causing the terms $p(a, b)$ and $p(c, b)$ to be perceived to be identical with respect to the model. A model with a domain size equal to the number of constants has the ability to distinguish between the constants (e.g.

$a = 0, b = 1, c = 2$) and thus the terms may evaluate to different truth values.

While these arguments do not hold in all cases, the PTTP+GLiDeS system is designed to run fully automatically and it has been found that, in the majority of cases, the domain size choice worked satisfactorily. Again, an informed user with knowledge of their problem can customize the domain size decisions to ones more appropriate to their particular situation.

The exception to the "domain size = number of constants" rule occurs when the problem contains equality. In the case of equality, the domain size is initially set to 2. MACE has inbuilt equality and will only allow $a = b$ if a and b are assigned to the same domain element. With the initial domain size set to either 2 (in the case of equality) or the number of constants, MACE is given some time to find models. If, at the end of this time, no models have been found then domain size setting resorts to the default MACE behaviour. The domain size is either i) incremented if the initial domain size was less than or equal to 2 or ii) set to 2 if greater than 2. This process is repeated until i) a model is found, ii) the time limit expires, or iii) the domain sizes between 2 and number of constants have been searched unsuccessfully.

5.3 Implementation and Performance

PTTP+GLiDeS performs best on non-Horn problems [4]. PTTP+GLiDeS uses MACE as its model generator. As discussed in Section 3.2, semantic guidance for linear deductions involving Horn sets has been researched previously. So experiments focused on non-Horn problems for the TPTP Library. MACE was used to generate up to 10 different models for each problem. The aim of this experiment was to evaluate the effectiveness of the model selection heuristic. It was felt gathering data for 10 models - comparing the heuristic value of each model with the model's performance with respect to guiding the prover to a solution - would provide sufficient data to either support or refute the assertion that the heuristic provides a means for selecting a model capable of guiding the theorem prover effectively.

5.3.1 Initial Implementation

The version of PTTP+GLiDeS used in the initial tests used clause ordering but no literal ordering (see Chapter 6, Section 6.2 for details). The “unbiased, non-Horn” problems from the TPTP Library v2.3.0 were used. The model generator was MACE v1.3.2.

There are 1208 unbiased, non-Horn CNF problems in the TPTP Library v2.3.0. For 1072 problems, no models were generated. Of the remaining 138 problems, MACE generated more than one model for 89 problems. For these 89 problems, MACE was used to generate up to 10 different models for each problem. PTTP+GLiDeS was then run on each problem using each of the different models generated by MACE, with a time limit of 300 seconds. In most cases, the choice of model made no real difference to the end result. Of the 89 problems with more than one model, for 22 of these problems MACE generated models that all scored the same. These problems had the same result whichever model was used (16 Timeout, 6 Theorem). For 67 problems, models that scored differently were generated. In 53 cases this made no difference to the result (35 Timeouts, 18 Theorems). For 14 problems the choice of model had a very significant effect (see Table 5.1). To show what effects the pruning has on the theorem provers results, the unguided PTTP was also run on the 22 problems of interest with a 300 second time limit. These results are shown next to the results for PTTP+GLiDeS.

As can be seen from Table 5.1, in 10 cases the model chosen was the best choice possible. In 2 cases, the model chosen was adequate - a proof was still found. In 2 cases, the worst model was chosen and no proof was found. The results of the unguided PTTP system are also shown. In 11 of the 14 cases, PTTP+GLiDeS with the best model performs as well or better than PTTP.

Table 5.1: Results of the best and worst models in the 14 cases where there was a significant difference in the outcomes.

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

	PTTP+GLiDeS					PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
CAT002-3	m01	44	theorem	555334	282.42	TIMEOUT		
	m02	44	TIMEOUT					
	* m03	40	theorem	535580	255.51			
	m04	40	theorem	599876	275.51			
	m05	40	theorem	553352	246.3			
	m06	40	theorem	596609	263.64			
	m07	40	theorem	534248	217.16			
	m08	40	theorem	602558	228.87			
	m09	40	theorem	556743	206.23			
	m10	40	theorem	604280	219.52			
KRS001-1	* m01	1	theorem	48	0.01	theorem	69	0.0
	m02	5	theorem	78	0.01			
	m03	5	TIMEOUT					
KRS002-1	* m01	4	theorem	631	0.1	theorem	334	0.02
	m02	8	theorem	273	0.03			
	m03	8	TIMEOUT					
KRS015-1	* m01	17	theorem	2310	0.32	theorem	555	0.03
	m02	21	TIMEOUT					
	m03	21	TIMEOUT					
MSC006-1	* m01	58	theorem	14308	5.25	theorem	1792624	100.47
	m02	61	TIMEOUT					
	m03	61	TIMEOUT					
	m04	64	TIMEOUT					
PUZ013-1	* m01	4	theorem	25	0.01	theorem	27	0.0
	m02	6	TIMEOUT					
	m03	5	TIMEOUT					
	m04	5	TIMEOUT					
PUZ014-1	* m01	4	TIMEOUT			theorem	127	0.06
	m02	6	theorem	117	0.02			
	m03	5	theorem	176	0.04			
	m04	5	theorem	267	0.05			

Table 5.1: (continued)

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

	PTTP+GLiDeS					PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
PUZ023-1	m01	101	TIMEOUT			theorem	100051	6.19
	m02	100	TIMEOUT					
	m03	100	TIMEOUT					
	m04	99	theorem	363774	124.86			
	* m05	98	theorem	101659	31.78			
	m06	99	TIMEOUT					
	m07	99	TIMEOUT					
	m08	100	TIMEOUT					
	m09	100	TIMEOUT					
	m10	100	TIMEOUT					
PUZ025-1	* m01	708	TIMEOUT			TIMEOUT		
	m02	712	TIMEOUT					
	m03	708	TIMEOUT					
	m04	708	TIMEOUT					
	m05	708	TIMEOUT					
	m06	712	TIMEOUT					
	m07	716	TIMEOUT					
	m08	712	theorem	296870	276.20			
	m09	712	theorem	296870	258.31			
	m10	712	theorem	296870	237.88			
SET006-1	m01	44	theorem	52	0.01	theorem	53	0.0
	m02	58	TIMEOUT					
	m03	44	theorem	60	0.01			
	* m04	36	theorem	51	0.0			
	m05	43	theorem	52	0.0			
	m06	54	TIMEOUT					
	m07	57	TIMEOUT					
	m08	37	theorem	60	0.01			
	m09	44	theorem	60	0.01			
	m10	36	theorem	51	0.0			
SYN055-1	* m01	2	theorem	87	0.01	theorem	37	0.0
	m02	3	TIMEOUT					

Table 5.1: (continued)

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

		PTTP+GLiDeS				PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
SYN084-2	m01	6	TIMEOUT			theorem	7934	0.56
	m02	5	TIMEOUT					
	m03	5	TIMEOUT					
	m04	5	TIMEOUT					
	m05	5	TIMEOUT					
*	m06	4	theorem	1225	0.13			
	m07	5	TIMEOUT					
	m08	4	theorem	1225	0.14			
	m09	5	TIMEOUT					
	m10	5	TIMEOUT					
SYN325-1	m01	4	TIMEOUT			theorem	5	0.0
	m02	4	TIMEOUT					
	m03	4	TIMEOUT					
	m04	4	TIMEOUT					
	m05	2	TIMEOUT					
*	m06	0	theorem	5	0.0			
	m07	0	theorem	5	0.0			
	m08	2	TIMEOUT					
	m09	4	TIMEOUT					
SYN352-1	m01	21	TIMEOUT			theorem	46191	4.01
	m02	20	TIMEOUT					
	m03	20	TIMEOUT					
	m04	20	TIMEOUT					
	m05	20	TIMEOUT					
	m06	19	TIMEOUT					
	m07	20	TIMEOUT					
	m08	20	TIMEOUT					
*	m09	17	theorem	12956	4.07			
	m10	20	TIMEOUT					

In the majority of cases were more than one model is generated, it seems to make little difference which model is used. If one model finds a solution, the rest will usually perform similarly. However in some cases, the choice of model has a dramatic effect of the end result. In these cases, the model selection criteria of selecting the model which produces

a ground clause set closest to semantic Horn seemed to be effective.

5.3.2 Final Implementation

After the initial implementation, some changes were made to the PTTP+GLiDeS prover. In addition to clause ordering, literal ordering was also included. It was felt that the model selection criteria tests should be rerun. The final testing was performed using v2.4.1 of the TPTP Library. The model generator used was MACE v1.3.3. The same conditions of a maximum of 10 models generated per problem and a time limit of 300 seconds were used.

There are 2744 non-Horn CNF problems in the TPTP Library v2.4.1. MACE failed to generate models for 2195 of these problems. Of the remaining 549 problems, MACE can generate more than one model for 358 problems. For these 358 problems, MACE was used to generate up to 10 different models for each problem.

In most cases, the choice of model made no real difference to the end result. Of the 358 problems with more than one model, for 89 of these problems MACE generated models that all scored the same. For all 89 problems with multiple models of equal score, all but 1 had the same result whichever model was used (55 Timeout, 25 Theorem, 9 No solution). For 256 problems, models that scored differently were generated. In 239 cases this made no difference to the result (146 Timeouts, 56 Theorems, 37 No Solutions). For 18 problems the choice of model had a very significant effect (see Table 5.2).

As can be seen from Table 5.2, in 7 cases the model chosen was the best choice possible. In 3 cases, the model chosen was adequate - a proof was still found. In 8 cases, the worst model was chosen and no proof was found.

Table 5.2: Results of the best and worst models in the 18 cases where there was a significant difference in the outcomes.

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

	PTTP+GLiDeS					PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
KRS015-1	m01	84	theorem	856	0.26	theorem	555	0.03
	m02	86	theorem	902	0.25			
	m03	84	NO SOLUTION					
	m04	86	NO SOLUTION					
	m05	84	NO SOLUTION					
	m06	88	NO SOLUTION					
	m07	84	theorem	865	0.20			
	* m08	78	theorem	902	0.25			
	m09	84	NO SOLUTION					
	m10	94	NO SOLUTION					
MSC006-1	* m01	58	theorem	14,308	5.24	theorem	1,792,624	100.47
	m02	61	TIMEOUT					
	m03	61	TIMEOUT					
	m04	64	TIMEOUT					
NUM016-1	m01	30	theorem	113	0.03	theorem	160	0.01
	m02	31	theorem	120	0.03			
	m03	31	theorem	109	0.03			
	m04	30	TIMEOUT					
	m05	31	TIMEOUT					
	m06	31	TIMEOUT					
	* m07	29	TIMEOUT					
	m08	30	TIMEOUT					
	m09	30	TIMEOUT					
	m10	29	theorem	126	0.03			
NUM016-2	m01	11	theorem	73	0.02	theorem	91	0.01
	m02	12	theorem	73	0.02			
	m03	11	theorem	73	0.01			
	* m04	10	NO SOLUTION					
	m05	11	NO SOLUTION					
	m06	10	NO SOLUTION					
	m07	10	NO SOLUTION					
	m08	11	NO SOLUTION					
	m09	10	NO SOLUTION					
	m10	11	theorem	73	0.02			

Table 5.2: (continued)

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

	PTTP+GLiDeS					PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
PUZ013-1	* m01	4	theorem	23	0.00	theorem	27	0.0
	m02	6	NO SOLUTION					
	m03	5	NO SOLUTION					
	m04	5	NO SOLUTION					
PUZ014-1	* m01	4	NO SOLUTION			theorem	127	0.06
	m02	6	theorem	114	0.02			
	m03	5	theorem	111	0.04			
	m04	5	theorem	212	0.05			
PUZ023-1	m01	101	TIMEOUT			theorem	100,051	6.19
	m02	100	TIMEOUT					
	m03	100	theorem	839,763	325.73			
	m04	99	theorem	102,469	32.32			
	* m05	98	theorem	101,437	31.12			
	m06	99	theorem	832,506	296.89			
	m07	99	TIMEOUT					
	m08	100	TIMEOUT					
	m09	100	TIMEOUT					
	m10	99	TIMEOUT					
PUZ035-1	* m01	350	TIMEOUT			TIMEOUT		
	m02	350	TIMEOUT					
	m03	352	theorem	45,246	22.77			
	m04	357	theorem	391,253	163.46			
PUZ035-2	* m01	350	TIMEOUT			TIMEOUT		
	m02	350	TIMEOUT					
	m03	352	theorem	74,038	36.81			
	m04	357	theorem	941,542	388.49			
SET006-1	m01	44	theorem	53	0.01	theorem	53	0.0
	m02	58	TIMEOUT					
	m03	44	theorem	61	0.01			
	* m04	36	theorem	51	0.00			
	m05	43	theorem	53	0.01			
	m06	54	TIMEOUT					
	m07	57	TIMEOUT					
	m08	37	theorem	55	0.01			
	m09	44	theorem	61	0.01			
	m10	36	theorem	51	0.01			

Table 5.2: (continued)

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

		PTTP+GLiDeS				PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
SET046-5	m01	4	NO SOLUTION			theorem	1,135	0.05
	m02	3	NO SOLUTION					
	m03	5	NO SOLUTION					
	m04	3	theorem	1,220	0.30			
	m05	5	theorem	385	0.05			
*	m06	2	theorem	805	0.16			
	m07	3	NO SOLUTION					
	m08	3	theorem	480	0.08			
SYN011-1	* m01	2	theorem	20	0.00	theorem	14	0.0
	m02	2	theorem	20	0.00			
	m03	2	NO SOLUTION					
SYN034-1	m01	4	NO SOLUTION			theorem	2,076	0.09
	m02	3	NO SOLUTION					
	m03	5	TIMEOUT					
	m04	3	theorem	1,204	0.29			
	m05	5	theorem	742	0.11			
*	m06	2	theorem	779	0.16			
	m07	3	TIMEOUT					
	m08	3	theorem	839	0.08			
SYN055-1	* m01	2	theorem	72	0.00	theorem	37	0.0
	m02	3	NO SOLUTION					
SYN069-1	m01	58	theorem	509	0.18	theorem	1,783	0.07
	m02	58	theorem	509	0.17			
	m03	58	theorem	509	0.17			
	m04	58	theorem	509	0.16			
	m05	58	theorem	509	0.15			
	m06	58	theorem	509	0.14			
	m07	58	theorem	509	0.14			
	m08	58	theorem	509	0.13			
	m09	58	theorem	509	0.12			
*	m10	55	NO SOLUTION					

Table 5.2: (continued)

*	Model selected by the Model Selection Heuristic
Scr	Total Excess True Literal Score
Infer.	Number of inference made during search
Time	CPU time (in seconds) for search for proof only

		PTTP+GLiDeS				PTTP		
Problem	Model	Scr	Result	Infer.	Time	Result	Infer.	Time
SYN325-1	m01	4	NO SOLUTION			theorem	5	0.0
	m02	4	NO SOLUTION					
	m03	4	NO SOLUTION					
	m04	4	NO SOLUTION					
	m05	2	NO SOLUTION					
	* m06	0	theorem	5	0.0			
	m07	0	theorem	5	0.0			
	m08	2	NO SOLUTION					
	m09	4	NO SOLUTION					
TOP001-2	m01	31	theorem	315,895	490.29	theorem	49,343	3.03
	* m02	25	TIMEOUT					
	m03	32	TIMEOUT					
	m04	31	theorem	315,895	426.13			
	m05	25	TIMEOUT					
	m06	32	TIMEOUT					
	m07	26	theorem	15,993	13.78			
	m08	32	theorem	315,888	343.46			
	m09	26	TIMEOUT					
	m10	32	TIMEOUT					
TOP005-2	m01	40	TIMEOUT			theorem	1,893,918	138.45
	m02	40	TIMEOUT					
	m03	40	theorem	2,475,591	192.71			
	m04	36	theorem	3,543,490	278.70			
	* m05	30	TIMEOUT					
	m06	37	TIMEOUT					
	m07	33	TIMEOUT					
	m08	41	TIMEOUT					
	m09	30	theorem	4,662,475	377.50			
	m10	36	theorem	3,543,490	279.04			

There are a number of differences between the results from the initial test (see Table 5.1) and those in the final testing (see Table 5.2). Some differences are due to the different test data (v2.3.0 unbiased nonHorn did not include problems NUM016-2, PUZ035-2, SET046-5, SYN011-1,TOP001-2, TOP005-2) and some is due to the difference in

MACE versions; either models were found where none had been found before (NUM016-1, PUZ035-1) or more then one model was found where only one have been found before (SYN034-1, SYN055-1, SYN069-1). Of more interest are the problems that are present in Table 5.1 and not in Table 5.2, such as CAT002-3.

In the initial testing only clause ordering was used and in the final testing literal ordering was added. For CAT002-3, in the initial testing some models produced a solutions and others did not. In the final testing, CAT002-3 is not present as under the new conditions all models produce a proof. This is also the case for KRS001-1, KRS002-1, PUZ025-1, SYN084-2 and SYN352-1. The models produced by MACE v1.3.2 and MACE v1.3.3 for these problems are the same. The difference in performance is the literal ordering. The models that produced solutions in the initial testing "got lucky" in that the default order of the literals favour these models. By adding literal ordering with respect to the model being used this unseen bias in the results is removed and all models perform equal well.

It would appear that the initial hypothesis that the model that produces a clause set closest to semantic Horn is best is false. Examining the results of the final testing would indicate that the model selection heuristic used here does not much perform significantly better than simply selecting the first model generated.

5.4 Some Examples

To try to understand what is happening with the semantic guidance, two problems from the final experiments were examined.

5.4.1 PUZ014-1 and PUZ013-1

For these problems, the total excess TRUE literal count heuristic resulted in a bad choice of model for PUZ014-1 and a good choice for PUZ013-1. Any model choice other then m01 would have produced a solution for PUZ014-1 and any model choice other than m01 would not have produced a solution for PUZ013-1. These two problems are from the

puzzle section of the TPTP library and are different attempts to prove the same problem. The problem stated in English, reads as follows (taken from header section of PUZ014-1):

> All the boys, in a certain school, sit together in one large room every evening. They are of no less than five nationalities - English, Scotch, Welsh, Irish, and German. One of the Monitors (who is a great reader of Wilkie Collins' novels) is very observant and takes MS. notes of almost everything that happens, with the view of being a good sensational witness, in case any conspiracy to commit a murder should be afoot. The following are some of his notes.
>
> 1. Whenever some of the English boys are singing "Rule, Britannia," and some not, some of the Monitors are wide awake.
> 2. Whenever some of the Scotch are dancing reels, and some of the Irish fighting, some of the Welsh are eating toasted cheese.
> 3. Whenever all the Germans are playing chess, some of the Eleven are not oiling their bats.
> 4. Whenever some of the Monitors are asleep, and some not, some of the Irish are fighting.
> 5. Whenever some of the Germans are playing chess, and none of the Scotch are dancing reels, some of the Welsh are not eating toasted cheese.
> 6. Whenever some of the Scotch are not dancing reels, and some of the Irish are not fighting, some of the Germans are playing chess.
> 7. Whenever some of the Monitors are awake, and some of the Welsh are eating toasted cheese, none of the Scotch are dancing reels.
> 8. Whenever some of the Germans are not playing chess, and some of the Welsh are not eating toasted cheese, none of the Irish are fighting.
> 9. Whenever all of the English are singing "Rule, Britannia," and some of the Scotch are not dancing reels, none of the Germans are playing chess.

10. Whenever some of the English are singing “Rule, Britannia,”, and some of the Monitors are asleep, some of the Irish are not fighting.
11. Whenever some of the Monitors are awake, and some of the Eleven are not oiling their bats, some of the Scotch are dancing reels.
12. Whenever some of the English are singing “Rule, Britannia,” and some of the Scotch are not dancing reels, ...

Here the MS. breaks off suddenly. The problem is to complete the sentence, if possible.

To convert this problem into FOL, the last rule is implemented as “if some of the English are singing “Rule, Britannia”, and some of the Scotch are not dancing reels, then are the monitors all awake?”. Using proof by contradiction it can either be assumed that “some of the monitors are awake” is FALSE or that “some monitors are not awake” is TRUE, and then search for a contradiction. PUZ013-1 uses the first option and PUZ014-1 uses the second. The models generated by MACE are the same for each as the set of model clauses (shown in Figure 5.3) are identical - it is only the conjecture that differs.

For the conjecture $\sim$`some_monitors_are_awake` that is used by PUZ013-1, the obvious approach is to attempt to use Rule 1. Rule 1 says that if `some_english_sing` and `some_english_sing_not` are TRUE then `some_monitors_are_awake` is TRUE. This is the approach taken by the unguided theorem prover and results in the proof shown in Figure 5.1.

Unfortunately, this proof is not possible with any of the models generated for the model clauses. The first model, $m01$, prunes this proof away very early as `some_english_sing_not` is FALSE in this model. All of the remaining models reject this proof much later as `some_monitors_are_not_awake` is FALSE in all models. These two pruning points are highlighted in Figure 5.1.

Another refutation for PUZ013-1 is possible and is found using $m01$. This refutation is shown in Figure 5.2. This refutation is a re-organization of the previous proof. The points that were pruned in Figure 5.1 have been moved to the leaf nodes in Figure 5.2. The

Rule 1	∼some_english_sing ∨ ∼some_english_sing_not ∨ some_monitors_are_awake
Rule 2	∼some_scotch_dance ∨ ∼some_irish_fight ∨ some_welsh_eat
Rule 3	∼some_germans_play ∨ some_germans_play_not ∨ some_of_the_eleven_are_not_oiling
Rule 4	∼some_monitors_are_awake ∨ ∼some_monitors_are_not_awake ∨ some_irish_fight
Rule 5	∼some_germans_play ∨ some_scotch_dance ∨ some_welsh_eat_not
Rule 6	∼some_scotch_dance_not ∨ ∼some_irish_fight_not ∨ some_germans_play
Rule 7	∼some_monitors_are_awake ∨ ∼some_welsh_eat ∨ ∼some_scotch_dance
Rule 8	∼some_germans_play_not ∨ ∼some_welsh_eat_not ∨ ∼some_irish_fight
Rule 9	∼some_english_sing ∨ some_english_sing_not ∨ ∼some_scotch_dance_not ∨ ∼some_germans_play
Rule 10	∼some_english_sing ∨ ∼some_monitors_are_not_awake ∨ some_irish_fight_not
Rule 11	∼some_monitors_are_awake ∨ ∼some_of_the_eleven_are_not_oiling ∨ some_scotch_dance
Rule 12a	some_english_sing
Rule 12b	some_scotch_dance_not
Auxiliary 1	some_english_sing_not ∨ some_english_sing
Auxiliary 2	some_monitors_are_not_awake ∨ some_monitors_are_awake
Auxiliary 3	some_scotch_dance ∨ some_scotch_dance_not
Auxiliary 4	some_irish_fight ∨ some_irish_fight_not
Auxiliary 5	some_welsh_eat ∨ some_welsh_eat_not
Auxiliary 6	some_germans_play ∨ some_germans_play_not

Table 5.3: Model clause for PUZ013-1 and PUZ014-1

Table 5.4: Four models generated by MACE for PUZ014-1

Model Number	Statements supported by the model
$m01$	None of the Scots are dancing reels. All of the Eleven are oiling their bats None of the Germans are playing chess. All of the English are singing. All of the Welsh are eating toasted cheese. All of the Irish are fighting. All monitors are awake.
$m02$	None of the Scots are dancing reels. All of the Eleven are oiling their bats. Some of the Germans are playing chess and some are not. Some of the English are singing and some are not. Some of the Welsh are eating and some are not. None of the Irish are fighting. All monitors are awake.
$m03$	None of the Scots are dancing reels. All of the Eleven are oiling their bats. Some of the Germans are playing chess and some are not. Some of the English are singing and some are not. None of the Welsh are eating toasted cheese. None of the Irish are fighting. All monitors are awake.
$m04$	Some of the Scots are dancing and some are not. Some of the Eleven are not oiling their bats. None of the Welsh are eating toasted cheese. All Germans are playing chess. Some of the English are singing and some are not. None of the Irish are fighting. All monitors are awake.

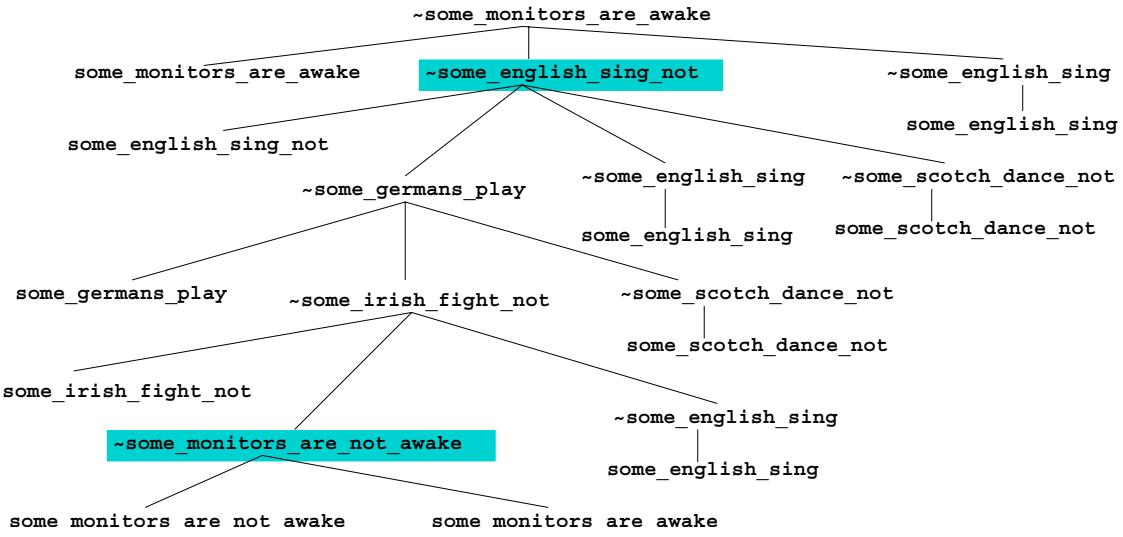

Figure 5.1: Tableau ME proof for PUZ013-1

other models, $m02$, $m03$ and $m04$, prune this proof as `some_english_sing_not` TRUE for these models.

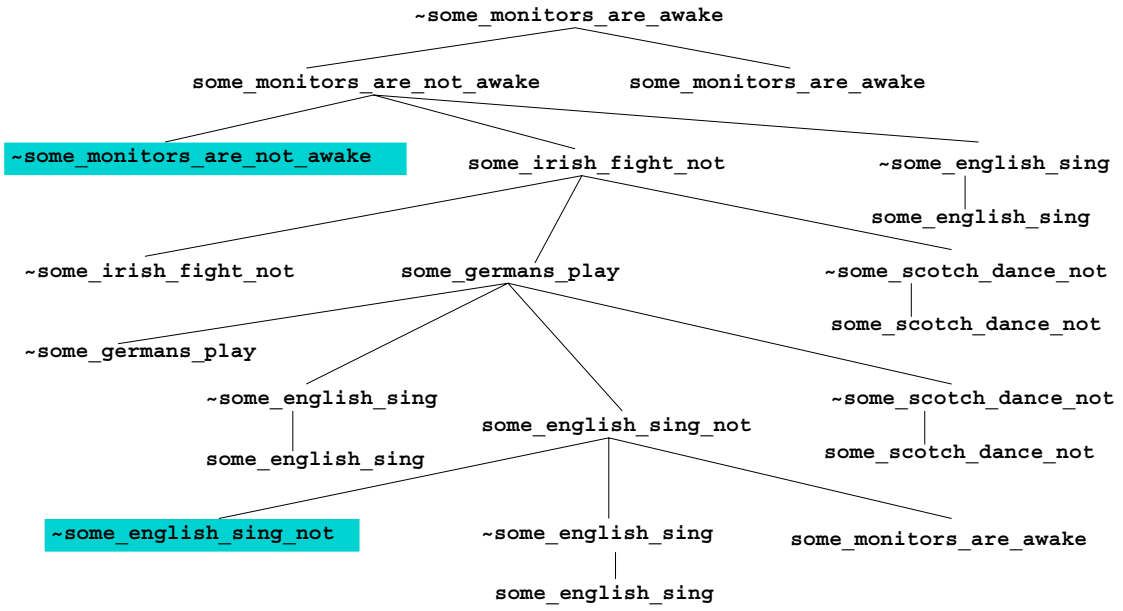

Figure 5.2: Tableau GLiDeS ME proof for PUZ013-1

The result for the four models is reversed when PUZ014-1 is attempted. In this problem the conjecture is `some_monitors_are_not_awake`. The refutation produced by the unguided prover is quite large and not compatible with any of the four models. For this

problem, unlike PUZ013-1, it is the models $m02$, $m03$ and $m04$ that produce solutions and $m01$ that does not.

As the models for both these problems are the same, it is perhaps not surprising that the model selection heuristic gets it right for one and not the other.

5.5 Summary

In the majority of cases where more than one model is generated, it seems to make little difference which model is used. If one model finds a solution, the rest will usually perform similarly. However in some cases, the choice of model has a dramatic effect of the end result. Initially it was hoped that the closer to semantic Horn a model makes a set of clauses the better the guidance. This does not generally seem to be the case. As it has been shown in Section 4.1.2 that the semantic pruning is complete only for models that yield an excess TRUE literal count of 0, if such a model is available it makes the best choice (see SYN325-1). Once the model has an excess TRUE literal score above 0, the value of this measure is limited.

Chapter 6

PTTP+GLiDeS Implementation and Performance

This chapter describes the implementation and performance of the PTTP+GLiDeS system. The initial implementation is discussed and its performance reviewed in Section 6.1. Additional features such as ordering of clauses and literals is discussed in Section 6.2. Finally the performance of the system against the different CNF categories in the TPTP library is examined.

6.1 Initial Implementation

PTTP+GLiDeS consists of modified versions of PTTP v2e [42] and MACE v1.3.3 [28], combined by a `csh` script. PTTP+GLiDeS takes problems in TPTP [44] format as input. The `tptp2X` utility is used to transform the input problem to PTTP and MACE formats. The transformation to PTTP format selects the first negative conjecture clause as the top clause for the linear deduction. If there aren't any negative conjecture clauses, then the first conjecture clause is selected.

A `perl` script is used to remove the first conjecture clause from the MACE format file, and MACE is called to generate a model of the remaining clauses. In this initial implementation MACE is only required to output a single model. If MACE is unable to

generate a model then PTTP+GLiDeS terminates. Otherwise MACE outputs its model in the form of Prolog facts, e.g.,

```
eval(domain,0).
eval(domain,1).
eval(functor,a,0).
eval(functor,b,1).
eval(functor,f(0),0).
eval(predicate,p(0,0),true).
```

These Prolog facts are interpreted as follows:

`eval(domain,0).`
: This fact states that one of the domain elements is `0`.

`eval(functor,a,0).`
: `a` is a functor of arity 0 that is mapped to the domain value `0`.

`eval(functor,f(0),0).`
: `f` is a unary functor and when given the argument `0` it returns the domain value `0`.

`eval(predicate,p(0,0),true).`
: `p` is a binary predicate symbol. `p(0,0)` is mapped to the truth value TRUE.

Once the model generator is finished the modified PTTP is run. The modified PTTP produces Prolog procedures that

i) maintain a list of all A-literals that have been produced in the deduction so far, and
ii) call a semantic checking procedure, `check` (see Appendix A), after each extension and reduction operation.

The facts produced by MACE are used to interpret the A-literals. The semantic checker pulls the term apart and replaces firstly constants and variables, and then functors with domain values, and finally predicates are given a truth-value. For example, the A-literal `p(f(a),a)` would be pulled apart to give `f(a)` and `a`. As the first term is not yet a constant or variable, it is pulled apart to give `a`. `a` is mapped to `0` by the fact `eval(functor,a,0)`.

Replacing a for its domain value gives f(0). Using the fact eval(functor, f(0), 0) allows f(a) to be replaced by 0. This enables is to replace p(f(a), a) by p(0, 0) and use the fact eval(predicate, p(0, 0), true) to assign the truth value of TRUE to the A-literal. If the semantic checking procedure finds an A-literal that is TRUE then the extension or reduction is rejected.

6.1.1 Performance

Testing has been carried out using 1248 "difficult, unbiased" problems from the TPTP library v2.4.1. The testing was done on a SUN Ultra-80, with a CPU time limit of 600 seconds. For PTTP+GLiDeS, MACE produced models for 437 of the 1248 problems and thus PTTP+GLiDeS could attempt only those problems. Table 6.1 gives an overall summary of the results.

For the problems solved, the CPU times taken and the number of inferences made during the search were recorded. For PTTP+GLiDeS, the number of rejected inferences was also recorded. The CPU time for PTTP+GLiDeS includes: time taken for preprocessing of the MACE input file to exclude the chosen top clause leaving only the model clauses; model generation and output time; and the Prolog search time. The CPU time for PTTP includes the Prolog search time only.

The number of inferences made during the search give an indication of the amount of the search space being covered during the search. A smaller inference count on the same problem would not necessarily indicate that the proof itself is any smaller but that less of the search space was covered before the proof was found.

Of the 437 problems for which models were generated, plain PTTP solved 38 and PTTP+GLiDeS solved 24. These 24 were a subset of the 38 solved by PTTP. In 23 of the 38 problems solved by PTTP, the models used by PTTP+GLiDeS made no difference to the search for a proof by PTTP+GLiDeS, i.e., there were no rejected inferences recorded. In these 23 cases, PTTP+GLiDeS performs the same search as PTTP but, with the overhead of the semantic checking, solves only 14 within the time-limit. The remaining

15 problems, i.e., those where the semantic guidance in the PTTP+GLiDeS system had some effect, are shown in Table 6.2.

The inferences columns give the total number of extension and reduction operations performed during the search for a solution. The rejected inferences are the number of inference operations that were rejected by the semantic pruning routine. The inference ratio shows the number of inferences made by PTTP+GLiDeS relative to PTTP.

PTTP+GLiDeS timed out on five problems that PTTP solved. In four of these five problems, PTTP+GLiDeS solves them eventually. For `TOP005-2` PTTP+GLiDeS prunes the PTTP solution and fails to find another. In all cases where both systems found a solution, PTTP+GLiDeS takes no more inferences than PTTP, and in many cases significantly less. The times taken by PTTP+GLiDeS are higher than for PTTP in most cases. Two interesting cases to note are `CAT012-3` and `GRP008-1`. These are non-Horn problems, and have the best reduction in inference counts. Of the 15 problems solved by either system, 8 are non-Horn and it is in these cases (with the exception of `TOP005-2`) that PTTP+GLiDeS performs best.

In the case of `TOP005-2` the model found by MACE is inadequate. The proof found by PTTP contains reductions but no complementary internal nodes across branches of the tableau. A model exists that will find this proof but unfortunately it is not generated by MACE. The model found by MACE in this case has all functors of arity 1 or greater mapped to the same domain element. This removes a lot of the detail from the model and results in the proof found by PTTP being rejected by PTTP+GLiDeS.

Total Number of Problems:	1248	
Number of Problems with Models:	437	
Number of Problems Solved from 437:	PTTP	PTTP+GLiDeS
	38	24
with Useful Models:	PTTP	PTTP+GLiDeS
	15	10

Table 6.1: Summary of experimental data.

Problem	PTTP		PTTP+GLiDeS			Inference Ratio
	CPU time (sec)	Inferences	CPU time (sec)	Inferences	Rejected inferences	
* CAT003-3	139.9	2,445,727	409.0	945,266	215,287	0.39
CAT004-4	125.4	3,339,787		TIMEOUT		
* CAT012-3	10.1	175,367	7.7	49,150	4,124	0.28
* CAT016-3	1.8	529	1.5	399	21	0.75
* CAT017-3	1.3	1,616	1.5	734	58	0.45
* GRP008-1	108.1	1,938,834	53.6	404,620	3,896	0.21
HEN009-5	55.4	1,122,032		TIMEOUT		
PLA007-1	16.7	342,934		TIMEOUT		
PLA016-1	9.3	176,746	466.1	176,746	710	1.0
PLA019-1	14.5	287,761		TIMEOUT		
PLA022-1	3.8	73,866	187.2	73,866	2	1.0
PLA022-2	1.3	2,485	108.5	2,485	2	1.0
* RNG040-2	2.7	32,127	8.5	30,274	758	0.94
* RNG041-1	1.6	6,325	2.7	5,935	207	0.94
* TOP005-2	139.2	1,893,918		TIMEOUT		
Average when solved by both Systems	27.94	485,362	127.01	168,948	22,507	

Table 6.2: Results for problems where semantic guidance rejected some inferences. Non-Horn problems are marked with "*".

6.2 Ordering of Clauses

For the problems tested in Section 6.1.1, the clauses were presented to PTTP+GLiDeS in the order in which they occurred in the TPTP library. Changes to the order in which clauses are presented may result in changes in the path taken through the search space. This means the performance of the system, as describe in Section 6.1, may depend on the format of the problem to be solved. This is undesirable. A system should be able to solve a problem consistently regardless of the order in which the clauses are presented. This means the system needs to perform its own ordering.

Ordering the clauses using semantic information has been achieved by the assigning of a measure of the "trueness" of each clause and then ordering the clauses with respect to this measure. Each clause is grounded with respect to the domain of the generated model. For each clause, the set of ground clauses are examined and a value of TRUE or FALSE is given to each literal. The total number of TRUE ground literals for each clause is calculated and divided by the total number of ground literals generated for that clause. This gives a number between 0 and 1 that rates the "trueness" of the clause. A clause gets the value 0 if all its literals are FALSE in the model for any instantiation. A clause gets the value 1 if all its literals are TRUE in the model for any instantiation. Literals within a clause can be ordered in the same manner.

Having obtained a measure by which to order the clauses and literals, the next question is which order. If clauses and literals are ordered in a ascending manner with respect to "trueness", the system is presented with clauses least likely to result in pruning first. If clauses and literals are ordered in a descending manner with respect to "trueness", the system is presented with clauses most likely to result in pruning first. In the first case, it would be expected that the clauses least likely to result in pruning are the ones most likely to lead to a proof, and so this would appear to be the better choice. Experiments were carried out to determine if this was the case. As it turns out, it appears that the sooner pruning occurs, the more dramatic the reduction in the search space and the more effective the pruning is.

Testing has been carried out using the same 1248 "difficult, unbiased" problems from the TPTP library v2.4.1 and under the same conditions as in Section 6.1.1. Table 6.3 shows a summary of the results. As can be seen from these results, either ordering obtains better results than no ordering at all with 3 more problems solved. Table 6.4 shows the results for all problems solved by any system. CPU time includes the time taken to generate models and order the clauses using the models if applicable.

Total Number of Problems:	1248		
Number of Problems with Models:	437		
Number of Problems Solved from 437:	No-ordering	Ascending	Descending
	24	27	27

Table 6.3: Summary of experimental data.

For some problems, changing the ordering has no impact on the search path. Problems `LCL194-1`, `LCL230-1` and `LCL231-1` are solved by all three methods in the same number of inferences. Some problems are solved with no rejected inferences, but different inference counts. For these problems, while the semantic pruning has had no impact, the semantic ordering is playing a role. `RNG001-5` is not solved without ordering, but with ordering (either ascending or descending) a proof is found.

While both the ascending and descending ordering methods solve an equal number of problems, overall the descending ordering results in lower averages for CPU time, number of inferences made, and number of rejected inferences. This seems to indicate that a few early rejected inferences have a greater impact on the guidance than more rejected inferences later in the search.

Table 6.4: Comparison of results using no ordering, ascending ordering, and descending ordering based on the "trueness" rating of each clause.

Problem	**No Order**			**Ascending**			**Descending**		
	CPU Time(s)	Infer.	Rej. Infer.	CPU Time(s)	Infer.	Rej. Infer.	CPU Time(s)	Inferences	Rej. Infer.
CAT001-3		TIMEOUT			TIMEOUT		499.5	629,924	21,937
CAT002-3		TIMEOUT			TIMEOUT		55.0	106,873	4,499
CAT003-3	432.1	945,266	215,287	408.5	947,246	215,635	6.1	11,074	530
CAT012-3	7.1	49,150	4,124	7.6	48,535	4,114	7.6	48,535	4,114
CAT012-4	2.7	16,356	0	3.2	15,705	0	3.2	15,705	0
CAT016-3	0.9	399	21	1.4	381	21	1.4	381	21
CAT017-3	0.9	734	58	1.5	716	58	1.5	716	58
GRP008-1	53.5	404,620	3,896	57.8	435,913	4,181	57.8	435,913	4,181
HEN009-5		TIMEOUT		354.3	377,437	29,234	354.3	377,437	29,234
LCL040-1	122.7	521,758	0	119.3	522,574	0	119.3	522,574	0
LCL064-1	3.8	13,831	0	4.3	13,462	0	4.2	13,462	0
LCL065-1	47.9	215,593	0	40.5	184,315	0	40.5	184,315	0
LCL194-1	11.0	460	0	11.5	460	0	11.6	460	0
LCL195-1	38.8	10,403	0	38.8	10,403	0	38.7	10,403	0
LCL230-1	273.2	58,295	0	267.3	58,295	0	268.5	58,295	0
LCL231-1	354.2	82,841	0	344.1	82,841	0	344.2	82,841	0
LCL359-1	83.5	1,404	0	84.1	1,167	0	84.0	1,167	0
PLA001-1	32.1	28,610	0	35.0	31,941	0	35.2	31,941	0
PLA016-1	489.9	176,746	710	378.3	142,434	537		TIMEOUT	
PLA019-1		TIMEOUT		594.0	231,976	944		TIMEOUT	
PLA022-1	189.7	73,866	2	168.4	61,272	3	206.9	96,916	0
PLA022-2	108.2	2,485	2	107.9	1,933	1	181.2	74,687	0

Table 6.4: (continued)

Problem	**No Order**			**Ascending**			**Descending**		
	CPU Time(s)	Infer.	Rej. Infer.	CPU Time(s)	Infer.	Rej. Infer.	CPU Time(s)	Inferences	Rej. Infer.
RNG001-5	TIMEOUT			562.1	3,926,803	0	565.1	3,926,803	0
RNG005-2	8.8	94	0	9.4	163	0	9.5	163	0
RNG006-2	8.1	249	0	8.7	423	0	8.7	423	0
RNG037-2	8.9	212	0	9.4	199	0	9.5	199	0
RNG040-1	1.0	130	0	1.7	236	0	1.8	236	0
RNG040-2	8.8	30,274	758	8.6	30,144	758	8.5	30,144	758
RNG041-1	2.2	5,935	207	3.5	6,971	441	3.5	6,971	441
Average when solv. by all	78.26	107,085	9,755	75.76	106,752	9,792	63.21	70,762	439
Average when solv. by ordered				106.36	270,381	10,178	94.93	237,270	1,573

6.2.1 Performance of GLiDeS+Model selection heuristic

Experiments were conducted to review the effect of adding the model selection heuristic to the existing GLiDeS system. In addition to the model selection heuristic, two additional changes were made to the system described in Section 6.2.

1. It has been observed that unless pruning occurs early in the proof search, the guidance it provides is minimal compared to the overhead of performing the semantic checks. Taking this into account, the current GLiDeS implementation turns off semantic pruning if no pruning has occurred in the first 3 search levels.

2. It has also been observed that for Horn problems without equality, no pruning occurs. So for Horn problems without equality, semantic ordering is applied but pruning is turned off.

Experiments were once more conducted using the 1248 “difficult, unbiased” problems from the TPTP library v2.4.1. A summary of these results are shown in Table 6.5. For those problems with models, PTTP+GLiDeS solves 40 problems, compared to 38 by PTTP. Eight of these problems were solved by both systems in the same number of inferences and so are of little interest.

Average total CPU time includes for PTTP+GLiDeS the time taken to generate the models, score and choose a model, and reorder the clauses and literals using the chosen model. Average CPU search time is the time taken for the search by the theorem prover only. For PTTP+GLiDeS this includes the time taken for the semantic checking if applicable. It is pleasing to note that, while the total CPU time average favours PTTP, the average CPU search time for PTTP+GLiDeS is significantly better than PTTP. This improvement is no doubt due to the selective use of the pruning. The overhead from semantic checks associated with the pruning is large unless the pruning is effective. By turning off the pruning when it is evident that it is having little effect and leaving the guidance to the semantic ordering alone appears to be the best option.

Results for those problem solved by either system where GLiDeS has some effect are shown in Table 6.6. PTTP solves one problem that PTTP+GLiDeS does not; this is TOP005-2 where the model selected by GLiDeS prunes away the solution found by PTTP. PTTP+GLiDeS solves 3 problems that PTTP does not. For two of these three problems, PTTP+GLiDeS doesn't perform any pruning. They are solved by the guidance provided by the reordering of the clauses and literals.

6.3 Performance of GLiDeS + Model selection heuristic across TPTP

The TPTP Library v2.4.1 contains 5882 problems, of which 4419 are CNF problems. These problems can be classified into Specialist Problem Classes (SPCs) [15]. There are 14 classifications in CNF, 7 pertaining to satisfiable problems and 7 pertaining to theorems. The two main classes of theorems are EPR (effectively propositional) and RFO (real first order). Effectively propositional problems are either true propositional

Total Number of Problems:	1248	
Number of Problems with Models:	437	
Number of Problems Solved from 437:	PTTP 38	PTTP+GLiDeS 40
Average Total CPU time for all problems solved by both systems:	43.06	89.57
Average CPU Search time for all problems solved by both systems:	42.41	31.62
Number of Problems Solved from 437 where GLiDeS had some effect:	30	32
Average Total CPU time for problems solved by either system where GLiDeS had some effect:	37.00	85.23
Average CPU Search time for problems solved by both systems where GLiDeS had some effect:	36.34	22.41

Table 6.5: Summary of experimental data.

Problem	PTTP CPU Time(s)	PTTP Infer.	PTTP Search Time(s)	PTTP+GLiDeS CPU Time(s)	PTTP+GLiDeS Infer.	PTTP+GLiDeS Rej. Infer.	PTTP+GLiDeS Search Time(s)	Infer. Ratio
CAT002-3		TIMEOUT		98.1	107,373	4,532	95.68	≈ 0.0
CAT003-3	145.0	2,445,727	144.26	10.2	11,081	590	7.82	0.005
CAT004-4	129.4	3,339,787	128.77	64.3	1,398,313	off	62.19	0.419
CAT012-3	9.8	175,367	9.08	8.2	48,535	4,114	6.10	0.277
CAT012-4	1.3	16,356	0.62	2.6	15,705	off	0.64	0.960
CAT016-3	0.7	529	0.02	2.1	381	21	0.03	0.720
CAT017-3	0.8	1,616	0.07	2.1	716	58	0.07	0.443
COL011-1	54.0	655,391	53.37	80.6	899,667	off	78.65	1.373
COL033-1	51.6	677,020	51.03	55.5	682,123	off	53.50	1.008
GRP008-1	111.5	1,938,834	110.87	57.3	435,913	4,181	55.30	0.225
HEN009-5	55.4	1,122,032	54.73	28.4	502,183	off	26.37	0.448
LCL014-1		TIMEOUT		390.7	4,890,578	-	388.44	≈ 0.0
LCL040-1	38.3	521,758	37.65	37.5	522,574	-	34.93	1.002
LCL042-1		TIMEOUT		558.5	10,883,088	-	555.76	≈ 0.0
LCL064-1	1.5	13,831	0.94	3.5	13,462	-	0.88	0.973
LCL065-1	14.4	215,593	13.77	14.0	184,315	-	11.49	0.855
LCL359-1	0.7	1,404	0.09	84.7	1,167	-	0.10	0.831
PLA001-1	1.7	28,610	1.00	11.4	31,941	-	1.11	1.116
PLA007-1	16.4	342,934	15.77	119.8	276,543	-	12.61	0.806
PLA016-1	8.9	176,746	8.24	113.9	142,434	-	6.62	0.806
PLA019-1	14.2	287,761	13.59	118.3	231,976	-	11.13	0.806
PLA022-1	3.3	73,866	2.62	109.4	61,272	-	2.13	0.830
PLA022-2	0.7	2,485	0.11	107.1	1,933	-	0.08	0.778
RNG001-5	207.2	4,777,282	206.46	171.6	3,926,803	-	167.87	0.822
RNG005-2	0.7	94	0.00	385.0	163	-	0.01	1.734
RNG006-2	0.7	249	0.01	10.9	423	-	0.02	1.699
RNG037-2	0.7	212	0.01	384.0	199	-	0.01	0.939
RNG040-1	0.7	130	0.01	4.1	236	off	0.02	1.815
RNG040-2	2.2	32,127	1.46	5.1	31,997	off	1.59	0.996
RNG041-1	1.0	6,325	0.31	3.0	10,054	off	0.60	1.590
SYN311-1	11.3	278,285	10.72	369.4	197,842	-	8.03	0.711
SYN312-1	188.9	4,716,889	188.31	107.6	2,496,242	-	99.92	0.529
TOP005-2	143.9	1,893,918	143.20		TIMEOUT			∞
Average when solv. by both	37.00	753,422.1	36.34	85.23	418,144.59		22.41	

Table 6.6: Results for problems solved by either PTTP or PTTP+GLiDeS where a model was generated and GLiDeS had some effect. "-" indicates no pruning attempted, "off" indicates pruning turned off after no early pruning occurred.

problems without variables or FO problems which have variables and constants but no functors of arity one or greater. These FO problems are termed *effectively* propositional as it is a simple exercise to instantiate the variables with all possible constant values thus yielding a ground clause set. So called "real" first order problems on the other hand contain functors of arity one or greater.

The PTTP+GLiDeS system works with unsatisfiable CNF problems and the SPCs that exist for these problems are:

THM_EPR_CNF - effectively propositional problems (414 problems)

THM_RFO_NEQ_CNF_HRN - real first order Horn problems containing no equality (396 problems)

THM_RFO_NEQ_CNF_NHN - real first order non-Horn problems containing no equality (438 problems)

THM_RFO_SEQ_CNF_HRN - real first order Horn problems, containing some equality (388 problems)

THM_RFO_SEQ_CNF_NHN - real first order non-Horn problems, containing some equality (1745 problems)

THM_RFO_PEQ_CNF_UEQ - real first order problems containing pure unit equality (431 problems)

THM_RFO_PEQ_CNF_NUE - real first order problems containing pure non-unit equality (125 problems)

Once again problems were tested with a 600s CPU time limit, on an UltraSPARC 80 Sun box. Model domain size was determined by the following set of conditions:

Equality MACE has built in equality. If we have two constants a and b such that $a = b$ should evaluate to TRUE then a and b have to map to the same domain element in the model. So if a clause set contains equality we start the search for a model with

a domain size 2 (MACE's default minimum) and progressively increment domain size until a model is found.

Number of constants If the problem contains no equality, then the domain size is set to be equal to the number of constants. The reason for this is that it was felt that in general the constants were meant to represent different objects and so should be assigned unique domain elements in the model to illustrate this. If a model cannot be found at this domain size, the domain size is reset to size 2 (MACE's default minimum) and the search for a model proceeds as for equality until the domain size once more reaches the number of constants. At this point the search for a model is abandoned.

	THM_CNF						
		RFO					
	EPR	NEQ		SEQ		PEQ	
		HRN	NHN	HRN	NHN	UEQ	NUE
No.Problems	414	396	438	388	1745	431	125
No.Problems with models	318	355	117	354	72	296	105
Number of Problems (with model) solved							
GLiDeS	247	190	58	89	13	68	11
PTTP	252	188	64	88	12	68	11
Number of Problems (with model) solved by one system but not the other							
GLiDeS/not PTTP	0	2	6	1	2	0	0
PTTP/not GLiDeS	5	0	13	0	1	0	0
Average Inferences for problems solved by both system							
GLiDeS	73,126	155,134	403,559	396,472	56,857	724,467	62,636
PTTP	137,486	177,469	472,278	439,318	449,052	595,657	118,424
Average Search CPU time for problems solved by both system							
GLiDeS	7.15	9.36	23.40	21.65	12.10	46.04	3.39
PTTP	7.21	11.78	24.05	22.58	27.92	40.88	6.42
Average Total CPU time for problems solved by both system							
GLiDeS	20.52	33.85	41.24	23.82	14.61	48.12	5.45
PTTP	10.62	12.11	24.41	22.99	28.35	41.21	6.75
Percentage of problems with model solved							
GLiDeS	77.67%	53.52%	48.74%	25.14%	18.06%	22.97%	10.48%
PTTP	79.25%	52.96%	56.30%	24.86%	16.67%	22.97%	10.48%

Table 6.7: Summary of results for PTTP and PTTP+GLiDeS across all 7 SPCs for unsatisfiable CNF problems from the TPTP Library v2.4.1

Results for the SPCs for both PTTP and PTTP+GLiDeS are shown in Table 6.7. As can be seen from this table, the GLiDeS guidance strategy has little impact on pure equality (PEQ) problems. Here PTTP and PTTP+GLiDeS solve the same problems in about the same times. Average inference counts for PTTP+GLiDeS are worse than PTTP for unit equality (UEQ) but better for non-unit equality (NUE).

For problems containing some equality (SEQ), PTTP+GLiDeS did better than PTTP on both Horn and non-Horn problems, solving more problems with better search times on average.

For problems with no equality (NEQ), PTTP+GLiDeS has mixed results. For the Horn problems, PTTP+GLiDeS was run without pruning - guidance was provided by the ordering only. Under these conditions, PTTP+GLiDeS did better than PTTP in number of problems solved, used marginally fewer average inferences and had better average search times. However the time taken to generate the models is significant and on average added approximately 24 seconds to the total CPU time, making PTTP+GLiDeS much slower overall.

For the non-Horn NEQ problems, PTTP solved more problems than PTTP+GLiDeS. While PTTP+GLiDeS solved 6 problems that PTTP did not, it failed to solve 13 problems that PTTP did. On the problems solved by both systems, PTTP+GLiDeS narrowly does better than PTTP in terms of average inferences and search time but again the addition of the model generation time means PTTP is faster on average then PTTP+GLiDeS.

A similar situation can be seen with the effectively propositional problems (EPR). PTTP solves more problems and while PTTP+GLiDeS used about half the number of inferences on average, PTTP was much faster with respect to average overall CPU time.

From these results, it can be seen that while the semantic guidance hasn’t had a dramatic effect, it has in most categories resulted in equal or better performance compared to PTTP.

Chapter 7

Conclusion

This chapter discusses how well the research objectives have been met and examines the performance of the PTTP+GLiDeS system. Areas for future work are also discussed.

7.1 Summary of objectives

This research aimed to discover a method for incorporating semantic guidance into linear deduction systems, in particular ME based linear systems. This has been achieved. The GLiDeS pruning strategy is a simple strategy of restricting the ME deduction to one where all A-literals are false in the guiding model. This can be easily incorporated into any ME based prover. The main issues of concern are with respect to the completeness of this strategy.

While Section 4.1.2 has shown that the GLiDeS strategy is complete for clause sets with a semantic Horn model, this situation arises infrequently. Whether GLiDeS can be shown to be complete in other cases is of interest but to some extent academic as it has been shown that GLiDeS is incomplete when combined with regularity pruning. As regularity pruning is such a powerful tool for ME deductions, in practise it would be unlikely to be “turned off” and so for practical use we know GLiDeS to be incomplete. The issue then is, how often does this problem arise? From experiments run on the SPCs in the TPTP library and examination of the results, 10 problems solved by PTTP and not by

PTTP+GLiDeS were not solved due to suspected incompleteness. By "suspected" it is meant that PTTP+GLiDeS pruned away the solution found by PTTP and failed to find another within the time limit. It may be possible that in some of these cases another solution might have been found eventually but that has not been established.

What has been shown is that given an effective model, the GLiDeS pruning and ordering works well. The number of inferences made during a search for a proof is a measure of the amount of the search space covered. The PTTP+GLiDeS system generally makes fewer inferences then PTTP. The results in Table 6.7 show that in the best case (THM_CNF_RFO_SEQ_NHN category problems), PTTP covered 8 times more of the search space than the GLiDeS guided system. Ordering alone (which is a complete strategy as it doesn't prevent any inferences, only changes the order in which they would occur) has proved to be an effective form of semantic guidance by itself, with the PTTP+GLiDeS system using ordering only solving more problems than PTTP in the THM_CNF_RFO_NEQ_HRN category. The key to semantic guidance is (as would be expected) the quality of the semantic provided. It would be expected that GLiDeS would be more effective when the semantics had been carefully considered before use.

Chapter 5 examined a heuristic for selecting models. This heuristic was shown not to be significantly more effective then simply accepting the first model discovered by the model generation software.

The final system, using semantic ordering of clauses and literals and selective use of semantic pruning is effective. By turning off the pruning when it is shown to be ineffective results in the pruning only being used when it will have an impact, so the overhead of semantic checks is only incurred when there is a likelihood of effective pruning. In Section 6.2.1, it was shown that the pruning was only used in 7 cases, and in most of these the reduction in the number of inferences made is dramatic. Only in one case (TOP005-2) did the pruning have a detrimental effect and in this case the selected model resulted in suspected incompleteness.

7.2 Future Work

It is possible to extend Restart ME [1] to include semantic guidance. Instead of restarting when a positive B-literal is encountered, restarts would be triggered by TRUE B-literals that cannot reduce. It is expected that a semantic Restart GLiDeS system would not suffer from incompleteness as regularity pruning is implemented in a restricted form. An early implementation of a semantic Restart GLiDeS system has shown this to be the case experimentally i.e. cases that are not solved by PTTP+GLiDeS are solved with PTTP+Restart+GLiDeS. Further work needs to be done on this interesting area.

Further examination of model generation and selection issues also need examination. Determination of the best domain size needs further examination but it may prove not to be possible to come up with some general setting that works for all problems. Also, excluding only the chosen top clause from the clause set to get the model clauses is not sufficient if the clause set is not minimally unsatisfiable. Additional work on selection of the model clauses may also assist with better model generation.

Bibliography

[1] P. Baumgartner and U. Furbach. Model Elimination Without Contrapositives. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 87–101. Springer-Verlag, 1994.

[2] P. Baumgartner and U. Furbach. Protein: A *pro*ver with a *t*heory *e*xtension *in*terface. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 769–773. Springer-Verlag, 1994.

[3] Peter Baumgartner and Ulrich Furbach. Consolution as a Framework for Comparing Calcui. *J. Symbolic Computation*, 16:445–477, 1993.

[4] M. Brown and G. Sutcliffe. PTTP+GLiDeS: Guiding Linear Deductions with Semantics. In N. Foo, editor, *Advanced Topics in Artifical Intelligence: 12th Australian Joint Conference on Artificial Intelligence, AI'99*, number 1747 in LNAI, pages 244–254. Springer-Verlag, 1999.

[5] A. Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, 1983.

[6] R. Caferra and N. Peltier. Model Building and Interactive Theory Discovery. In *Proceeding of Tableaux'95*, LNAI 918, pages 154–168. Springer, 1995.

[7] C-L. Chang. The Unit Proof and the Input Proof in Theorem Proving. *Journal of the ACM*, 17(4):698–707, 1970.

[8] C-L. Chang and R. C-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York and London, 1973.

[9] Seungyeob Choi and Manfred Kerber. Semantic Selection for Resolution in Clause Graphs. In Bob McKay and John Slaney, editors, *AI2002: Advances in Artificial Intelligence, 15th Australian Joint Conference on Artificial Intelligence, Canberra, Australia, December 2002, Proceedings*, number 2557 in LNAI, pages 83–94. Springer-Verlag, 2002.

[10] H. Chu and D. Plaisted. Semantically Guided First-order Theorem Proving using Hyper-linking. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in LNAI, pages 192–206. Springer-Verlag, 1994.

[11] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7), July 1962.

[12] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, July 1960.

[13] David Duffy. *Principles of Automated Theorem Proving*. John Wiley and Sons, 1991.

[14] Melvin Fitting. *First-Order Logic and Automated Teorem Proving*. Grduate Texts in Computer Science. Springer-Verlag, New York, 2nd edition, 1996.

[15] M. Fuchs and G. Sutcliffe. Homogeneous sets of atp problems. In *Proceedings of teh 15th Florida Artificial Intelligence Research Symposium (Pensecola, USA)*, pages 57–61, 2002.

[16] L.J. Henschen and L. Wos. Unit Refutations and Horn Sets. *Journal of the ACM*, 21(4):590–605, 1974.

[17] Kahil Hodgson and John Slaney. System description: Scott-5. In *Automated Reasoning: First International Joint Conference, IJCAR 2001 Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083/2001 of *LNCS*, pages 443–447. Springer-Verlag Heidelberg, 2001.

[18] Kahil Hodgson and John Slaney. TPTP, CASC and the development of a semantically guided theorem prover. *AI Communications*, 15(2-3):135–146, 2002.

[19] R. A. Kowalski. Predicate logic as a programming language. In *IFIP 74*, pages 569–574, 1974.

[20] R.A. Kowalski and D. Kuehner. Linear Resolution with Selection Function. *Artificial Intellience*, 2:227–260, 1971.

[21] Reinhold Letz and Gernot Stenz. *Handbook of Automated Reasoning*, chapter 28. Elsevier Science Publishers, 1999.

[22] D.W. Loveland. Mechanical Theorem Proving by Model Elimination. *Journal of the ACM*, 15(2):236–251, 1968.

[23] D.W. Loveland. A Simplified Format for the Model Elimination Theorem-Proving Procedure. *Journal of the ACM*, 16(3):349–363, 1969.

[24] D.W. Loveland. A Linear Format for Resolution. In Laudet M. et al., editor, *Proceedings of the IRIA Symposium on Automatic Demonstration*, pages 147–162. Springer-Verlag, 1970.

[25] D.W. Loveland. Near-Horn Prolog. In Lassez J-L., editor, *Proceedings of the 4th International Conference on Logic Programming*, pages 456–469. MIT Press, 1987.

[26] D. Luckham. Some Tree-paring Strategies for Theorem Proving. *Machine Intelligence*, 3:95–112, 1968.

[27] D. Luckham. Refinement Theorems in Resolution Theory. In Laudet M. et al., editor, *Proceedings of the Symposium on Automatic Demonstration*, pages 163–190. Springer-Verlag, 1970.

[28] W.W. McCune. A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, USA, 1994.

[29] W.W. McCune. MACE: Models and Counterexamples. http://www.mcs.anl.gov/home/mccune/ar/mace/, 1998.

[30] B Meltzer. Theorem-Proving for Computers: Some Results on Resolution and Renaming. *The Computer Journal*, 8:341–343, 1966.

[31] M.S Paterson and M. Wegman. Linear Unification. *Journal of Computer and Systems Science*, 16:158–167, 1978.

[32] D.A. Plaisted. Non-Horn Clause Logic Programming Without Contrapositives. *Journal of Automated Reasoning*, 4(3):287–325, 1988.

[33] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[34] J.A. Robinson. Automatic Deduction with Hyper-Resolution. *Intern. Journal of Computer Math.*, 1:227–234, 1965.

[35] P. Roussel. *Prolog : Manual de Reference et d'Utilisation*. Groupe d'Intelligence Artificielle, Marseille-Luminy, September 1975.

[36] J.R. Slagle. Automatic Theorem Proving with Renamable and Sematic Resolution. *Journal of the ACM*, 14:687–697, October 1967.

[37] J. Slaney. FINDER: Finite Domain Enumerator System Description. Technical Report Technical Report TR-ARP-2-94, Automated Reasoning Project, Australian National University, Canberra, Australia, 1994.

[38] J.K. Slaney. SCOTT: A Model-Guided Theorem Prover. In R. Bajcsy, editor, *Proceedings of the 13th International Conference on Artificial Intelligence*, pages 109–114. Morgan-Kaufman, 1993.

[39] J.K. Slaney and K. Hodgson. System description: Mscott. Technical Report TR-ARP-06-98, Automated Reasoning Project, ANU, 1998.

[40] M.E. Stickel. A Prolog Technology Theorem Prover. In *Proceedings of the 1st International Symposium on Logic Programming*, pages 211–217. IEEE Computer Society Press, 1984.

[41] M.E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4(4):353–380, December 1988.

[42] M.E. Stickel. A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. Technical Report Technical Note 464, SRI International, Menlo Park, USA, 1989.

[43] G. Sutcliffe. Linear-Input Subset Analysis. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in LNAI, pages 268–280, Saratoga Springs, NY, USA, June 1992. Springer-Verlag.

[44] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[45] L. Wos, D. Carson, and G.A. Robinson. The Unit Preference Strategy in Theorem Proving. In *Proceedings of the AFIPS 1964 Fall Joint Computer Conference* , pages 615–621. Spartan Books, 1964.

[46] Hantao Zhang. SATO: an Efficient Propositional Prover. *Proceedings of the 14th International Conference on Automated Deduction*, 1997.

[47] J. Zhang and H. Zhang. SEM: a System for Enumerating Models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI95)*, August 1995.

Appendix A

Semantic Checking

```
%-------------------------------------------------------------
%-- test_model\2
%-- Take the TopClause from the input clause set and pass to
%-- test_model\1
test_model((_ModelClauses,TopClause)):-
        !,
        test_model(TopClause).

%-------------------------------------------------------------
%-- test_model\1
%-- Negate the literals in the TopClause and check them
%-- against the model. All literals need to be FALSE in the
%-- model for the model to be useful. If they fail the check,
%-- print error message and exit with code 15
test_model(Y):-
        Y =.. [:-,query,Lits],
        split_and_reverse(Lits,CheckList),
        \+ check(CheckList,_),!,
        write('Centre clause not false in model'),
        nl,
        write(CheckList),
        nl,
        halt(15).

%-- otherwise, succeed.
test_model(_).

%-------------------------------------------------------------
%-- check/2
%-- Check list of A-literals against model
```

```
%-- Input Parameter: InAList - list of A-literals
%-- Ouput Parameter: OutAList - list of A-literals with
%--                              ground A-literals removed

%-- if model is a positive trivial model then use
%-- simplified p_check
check(InAList,OutAList):-
        model(pTrivial),!,
        p_check(InAList).
        remove_ground(InAList,OutAList).

%-- if model is a negative trivial model then use
%-- simplified n_check
check(InAList,OutAList):-
        model(nTrivial),!,
        n_check(InAList),
        remove_ground(InAList,OutAList).

%-- otherwise, take a copy of the A-list and perform full check
check(InAList,OutAList):-
        copy_term(InAList,CheckList),
        do_check(CheckList),
    !.

%-- if all fail then A-list doesn't pass the test
%-- increment the number of rejected inferences and fail
check(_,_):-
        !,
        inc_nrej,
        fail.

do_check([]).

do_check([H|T]):-
        is_false(H),
        do_check(T).

is_false(equal(X,Y)):-
        !,
        \+ eval_term(predicate,equal(X,Y),true).

is_false(pttpnot_equal(X,Y)):-
        !,
        eval_term(predicate,pttpnot_equal(X,Y),false).

is_false(Term):-
```

```
        eval_term(predicate,Term,false).

eval_term(predicate,equal(X,Y),true):-
        eval_args([X,Y],[Z,Z]),
        !.

eval_term(predicate,equal(_,_),false):-
        !.

eval_term(predicate,pttpnot_equal(X,Y),false):-
        eval_args([X,Y],[Z,Z]),
        !.

eval_term(predicate,pttpnot_equal(_,_),true):-
        !.

eval_term(Type,X,EX):-
        X =.. [F|Args],
        ( Args == [] ->
                EArgs = [];
          % or
                eval_args(Args,EArgs)),
        X2 =.. [F|EArgs],
        eval(Type,X2,EX).

eval_args([],[]).

eval_args([Head|Tail],[NewHead|NewTail]):-
        ( var(Head) ->
                !,
                eval(domain,Head),
                NewHead = Head;
                ( integer(Head) ->
                        NewHead = Head;
                eval_term(functor,Head,NewHead))),
        eval_args(Tail,NewTail).


split_and_reverse((X,Y),[NewX|Rest]):-
        !,
        reverse_sign(X,NewX),
        split_and_reverse(Y,Rest).
split_and_reverse((Y),[NewY]):-
        reverse_sign(Y,NewY).

reverse_sign(X,Not_X):-
```

```
        X =.. [Functor|Args],
        name(Functor,F),
        name(pttpnot_,Neg),
        ( append(Neg,F2,F) ->
                name(NewF,F2),
                Not_X =.. [NewF|Args];
          %true ->
                append(Neg,F,F2),
                name(NewF,F2),
                Not_X =.. [NewF|Args]).

n_check([]):- !.

n_check([Head|_InAList]):-
        Head =.. [Predicate|_],
        name(Predicate,P),
        name(pttpnot_,N),
        append(N,_,P),!,
        inc_nrej,
        fail.

n_check([_Head|AList]):-
        n_check(AList).

p_check([]):- !.

p_check([Head|AList]):-
        Head =.. [Predicate|_],
        name(Predicate,P),
        name(pttpnot_,N),
        append(N,_,P),!,
        p_check(AList).

p_check(_):-
        !,
        inc_nrej,
        fail.
```