# Resolution Based Techniques for Automated Proving of Theorems in Tarskian-Euclidian Geometry

**Sergei Savchenko**
School of Computer Science
McGill University, Montreal

October 1999

A Thesis Submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of Master of Science in Computer Science

0-612-64447-2

# Abstract

The discipline of automated theorem proving encompasses techniques which allow us to find a justification of a logical statement expressing an assertion in some domain of knowledge. Beside obvious importance for mathematics, many of the tasks traditionally associated with human intellect can be solved through application of these techniques. Methods based on Robinson's resolution form one of the cornerstones of automated theorem proving. The efficiency of these methods, however, is less than admissible for many interesting domains of mathematics. By studying the underlining axioms of the domain it is often possible to find some computational shortcut. This thesis overviews available generic methods and then considers possible refinements aimed at theorems in Euclidian geometry formulated on the Tarskian axiom system.

# Résumé

La discipline de démonstration automatique de théorèmes entoure des techniques ce qui laissent trouver une justification d'une phrase logique exprimant une affirmation dans un certain domaine de la connaissance. Près de l'importance évidente pour des mathématiques, plusieurs des tâches qui sont traditionnellement associées à l'intellect humain peuvent être résolues par l'application de ces techniques. Les méthodes basées sur la résolution de Robinson sont parmi les plus importantes pour le démonstration automatique de théorèmes. L'efficacité de ces méthodes, cependant, est moins qu'admissible pour beaucoup de domaines intéressants des mathématiques. En étudiant les axiomes soulignants du domaine, il est souvent possible de trouver certain accélération de calcul. Cette thèse fait une revue de méthodes principales pour démontrer des théorèmes et alors considère les améliorations possibles pour les théorémes dans la géométrie euclidienne formulée sur le système d'axiome de Tarski.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I'd like to express my profound gratitude to professor Monroe Newborn, my thesis supervisor, for introducing me to the subject, suggesting the topic and giving many valuable advises.

I am deeply indebted to my parents Nikolay Savchenko and Yanna Laevski for their support and advises which greatly motivated me throughout the life and in this undertaking in particular.

I'd like to thank professor Prakash Panangaden and professor Lorie Hendren for giving me an opportunity to teach introductory Computer Science courses which proved to be a tremendously valuable experience.

Thanks to the external reviewer, professor Bruce Spencer, for his valuable comments.

Many thanks to the faculty, staff and students of McGill University, School of Computer Science for creating such an excellent environment for studies and research.

# Chapter 1

# Introduction

Formal logic occupies an important place in the foundations of modern mathematics. It is used for reconstructing and analyzing mathematical proofs in a formal manner. One of the main tasks of formal logic is to provide efficient means to justify a theorem (i.e.: a conclusion) of some premises by a procedure carried out manually or, in a modern setting, by a computing device. Beside logic's importance for mathematics, many of the tasks traditionally associated with human intellect can be expressed as applications of formal logic and particularly of theorem proving. As a consequence, a mechanical procedure to prove theorems implemented and carried out by a computer can serve as an instrument to solve multiple mathematical as well as applied problems and tasks.

The desire to find a general decision procedure to prove theorems dates back at least to Leibniz (1646-1716), most probably going as far back as Aristotle's logic of syllogisms. The stronger interest to this subject emerged in the last century with the work of Boole on algebra of logic. The advances began truly in earnest, however, only around the beginning of this century with the axiomatizations for arithmetic and projective geometry by Peano, discovery of paradoxes of set theory and later with contributions by Hilbert and his school of logic.

As a consequence of early successes, Hilbert proposed a program for formalization of all of mathematics with axiomatized logic serving as a common base to demonstrate consistency of particular theories. These hopes were soon dashed, however, by Gödel's incompleteness results. The second of Gödel's incompleteness theorems states that in any consistent formal theory containing arithmetic, the sentence asserting the consistency of the theory itself is not provable within that theory.

Despite Gödel's discovery, by 1930s a very important theorem was proven by Herbrand opening a way for actual mechanical methods to prove theorems. Few years later Gentzen defined a natural notion of formal proofs which is closer to mathematical deduction than the Hilbert-type systems. Both Herbrand and Gentzen investigated the structure of mathematical proofs, whereas Gödel results were directed at provability [Le97].

In 1936 Church and Turing, strengthening Gödel's results, independently showed that, in fact, there cannot be a general decision procedure to check validity of formulas of first-order logic and thus any imaginable proof procedure for this logic has a weaker power – it can only show validity of formulas which are indeed valid. That result was demonstrated earlier in Gödel's completeness theorem which stated that any valid formula is provable. Church-Turing result however demonstrated that for invalid formulas the proof procedure may not terminate [Me64], [CL73].

With the invention of digital computers the interest for mechanical procedures to prove theorems was regained again and implementations for various mechanical methods started to emerge. By 1960 Herbrand's procedure was implemented by Gilmore and later a more efficient algorithm was introduced by Davis and Putnam. By 1965 a major contribution by Robinson gave a significantly more efficient algorithm. Since that time, many refinements and distinct methods appeared. Automated provers which were initially capable to prove only simple theorems grew in sophistication to the degree when some open mathematical problems were solved with their help, such as recent (1996) prove by *Argonne National Lab's* prover *EQP* the theorem that every Robbins algebra is in fact a Boolean algebra [Mc97].

Mechanical theorem proving methods also branched into multiple applied areas such as program verification, circuit design, expert systems and databases [Sa98].

Current advances in computer technologies, especially that in parallel architectures and fast networks connecting multiple microcomputers, promise greater efficiency of implementations for mechanical theorem provers in the future and hence these techniques will undoubtedly find even more uses then there are today.

We begin the thesis by reviewing principles of propositional and predicate logic. Whereas it is the latter which is of practical interest, the former allows us to formulate many concepts in a much more intuitive form. Chapter 3 is dedicated to fundamental principals of automated theorem proving. It is shown that there is a semantic and deductive approaches. A version of Herbrand's theorem is discussed and a procedure of semantic tree building is introduced. Further we turn to the deductive method of resolution and demonstrate that its completeness is implied by Herbrand's theorem. Whereas these procedures can do the job, they suffer from computational inefficiency. The rest of the thesis discusses various ways how to improve performance of resolution based theorem provers. Particularly, Chapter 4 details multiple general strategies. Although these, properly implemented, may give a considerable performance improvement, it is possible to go even further by concentrating on one class of theorems and exploiting special properties of their common axioms. Thus, Chapter 5 discusses specialized approaches for theorems in *Euclidian geometry*, more particularly those using axiom system formulated by Tarski. This class of theorems was chosen as it is quite representative of relatively difficult theorems one would want to proof automatically. Their difficulty comes from relatively high number of clauses, heavy use of equality predicate as well as of highly flexible axioms describing behavior of such predicates as *equidistance*, *betweenness* and *colinearity*. Several approaches will be proposed and heuristic observations made for efficient proving of geometry theorems.

Most of the observations and results which will be presented were obtained from experiments with *GLIDE* (Geometry Linear Iterative Deepening Engine) theorem prover written by the author and based on the experience of *TGTP* (The Great Theorem Prover) theorem prover by Professor M. Newborn [Ne97]. GLIDE prover incorporates some special refinements aimed at theorems with equality and more particularly at geometry theorems. The implementation and results of experiments (including performance comparisons with two popular provers) will be described in Chapter 5.

# Chapter 2

# Logic calculi

As the name "theorem proving" suggests, the primary interest of this discipline is giving a justification to logical sentences expressing theorems. Axiomatic method, allowing to solve this problem, has received a wide acceptance in mathematics. With this method we first formulate a system of laws which we accept without a proof for some domain. These laws are called *axioms*. Other laws, the *theorems*, can then be proven to be consequences of the axioms and thus valid (on the assumption that the axioms were).

Formalisms to express the axiom systems are referred to as *logic calculi* (such as *propositional calculus* or *predicate calculus*).

The logic calculi could be expressed as axiom systems themselves (with the help of *logic* axioms) and they are used as foundations to build axiom systems for other, particular domains which can then be treated as single formulas within the axiom system of a logic calculus. [Me64], [Sh67].

Thus, a logic calculus can be formulated to consist of a *language*, used to construct formulas and a way to define a theorem. This can be done with the help of *logic axioms* and the *rules of inference*.

A language consists of *symbols*. Any finite sequence of symbols is called an *expression* in that language. The meaningful expressions of the language are called *well-formed formulas* or simply *formulas*. Axioms must be formulas in the language of the logic calculus.

Rules of inference state under what conditions one formula, called the conclusion of the rule, can be inferred from other formulas called the hypotheses.

The *theorems* of an axiom system are either axioms or the conclusions of the inference rules whose hypotheses were theorems themselves.

A *proof* of a theorem is a finite sequence of formulas ending with the theorem being proven. The formulas in the sequence can either be axioms or the conclusions of some inference rules (hence theorems themselves) whose hypotheses precede that formula in the proof.

Beside such purely syntactic or deductive treatment of logic, a semantic or model-theoretic approach is also possible. With it, we directly define the meaning of certain elements of the language (e.g.: that of logical connectives (*see Section 2.1*) and enable interpretation of formulas. We will then verify validity of formulas based on the result of interpretations.

From the semantic view-point a formula will be *valid* if it will be considered *true* under all possible interpretations.

Both deductive and semantic approaches will serve as foundation for different mechanical procedures to prove theorems.

Particular logic calculi differ primarily in their language and hence in the formulas which can be expressed by the calculus. As a result, different logic calculi have different power to describe particular domains.

## 2.1 Propositional calculus

The *propositional* or *sentential calculus* is a formalism to express relatively simple axiom systems. It deals with declarative sentences, the propositions, which can be either *true* or $false$ but not both. The *true* or $false$ assigned to a proposition is called the *truth value* of the proposition.

The *language* of the propositional calculus consists of a countably infinite set of symbols (i.e.: $\{A, B, C, \ldots\}$) representing basic (or *atomic*) propositions augmented by a pair of parentheses and a small finite set of logical connectives (i.e.: $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$). The latter are used to express propositions where several atomic propositions are connected into a compound sentence.

**Definition 1** *A* **formula** *of propositional calculus is:*

- *an atomic proposition.*
- *If $A$ is a formula, then so is $(\neg A)$.*
- *If $A$ and $B$ are formulas then so are $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$.*

Clearly, some expressions are not formulas (e.g.: $(\Leftrightarrow A \vee))$. Although, the expressions $A \wedge B$ or $A \vee \neg B$ are also not formulas in a strict sence, it is customary to allow neglecting parentheses. Possible confusion can be avoided by instituting precedence ordering "$\succ$" upon the connectives of how tightly they bind symbols in the formula:

$$\neg \succ \wedge \succ \vee \succ \Rightarrow \succ \Leftrightarrow$$

thus, "$\neg$" has the highest binding priority and "$\Leftrightarrow$" the lowest.

**Example 1** *The formula $A \Rightarrow \neg B \wedge C$ will mean $(A \Rightarrow ((\neg B) \wedge C))$ when using the precedence ordering of the connectives given above.*

### 2.1.1 Semantics of propositional calculus

Let's consider the semantic side of propositional calculus.

An assignment of truth values to individual propositions or atoms of a formula is called an *interpretation* of that formula.

The truth value of a compound proposition depends on the truth values of the atomic propositions it consists of and the defined meaning of the logic connectives. The meaning of "$\neg$" *negation*, "$\wedge$" *conjunction*, "$\vee$"

*disjunction*, "$\Rightarrow$" *implication* and "$\Leftrightarrow$" *equivalence* connectives is defined as follows:

- $\neg A$ is *true* if $A$ is *false*. It is *false* if $A$ is *true*.
- $A \wedge B$ is *true* if both $A$ and $B$ are *true*. It is *false* otherwise.
- $A \vee B$ is *true* if either $A$ or $B$ is *true*. It is *false* if both $A$ and $B$ are *false*.
- $A \Rightarrow B$ is *false* if $A$ is *true* and $B$ is *false*. It is *true* otherwise.
- $A \Leftrightarrow B$ is *true* if $A$ and $B$ have the same truth values. It is *false* if $A$ and $B$ have different truth values.

The meanings of the connectives can be conveniently represented by a *truth table* (*see Table 2.1*).

| $A$ | $B$ | $\neg A$ | $A \wedge B$ | $A \vee B$ | $A \Rightarrow B$ | $A \Leftrightarrow B$ |
|---|---|---|---|---|---|---|
| *true* | *true* | *false* | *true* | *true* | *true* | *true* |
| *true* | *false* | *false* | *false* | *true* | *false* | *false* |
| *false* | *true* | *true* | *false* | *true* | *true* | *false* |
| *false* | *false* | *true* | *false* | *false* | *true* | *true* |

Table 2.1: Truth table of the connectives.

Whereas the meanings of negation, conjunction and disjunction have a clear and intuitive parallel with the everyday life, [1] that of implication and equivalence is less obvious. Implication connective attempts to capture the meaning of casuality of the type *'if ... then ... '*. Thus the sentence $A \Rightarrow B$ cannot be *true* when $A$ is *true* but $B$ is *false*. Similarly, the equivalence

[1]Natural languages posses two notions of disjunction: "inclusive or" – $A$ or $B$ or both and "exclusive or" – $A$ or $B$ but not both. Our meaning of disjunction coinsides with inclusive or, whereas the exclusive or can be modeled as $(A \vee B) \wedge \neg(A \wedge B)$.

connective attempts to capture the meaning of casuality of the type *'if and only if ... then ... '*. Although it is arguable to what extend implication and equivalence connectives capture the sense of the everyday life they are certainly convenient in the formal sense to express relationships between propositions.

A formula is *true* under an interpretation if it is evaluated to the logical value of *true* in that interpretation.

A formula which is true under all interpretations is called *valid* or a *tautology*. A formula which is false under all interpretations is called *inconsistent*, *unsatisfiable* or a *contradiction*. A formula which is true under some interpretations is called *consistent* or *satisfiable*.

An interpretation of a formula, under which that formula is true is called a *model* of the formula. An interpretation of a formula, under which that formula is false is called a *countermodel* of the formula.

Two formulas are said to be *logically equivalent* if and only if they have the same truth values under all interpretations. We will denote logical equivalence of formulas $A$ and $B$ by $A \equiv B$.

We can establish whether or not two formulas are logically equivalent by analyzing their truth tables. The truth tables essentially list truth values under all possible interpretations and thus if the truth tables will match, the formulas must be logically equivalent. Knowing pairs of logically equivalent formulas enable various transformations which preserve logical properties.

Using the technique of truth tables, it is not hard to verify that the following equivalence laws hold. Thus for any formulas $A$, $B$ and $C$ we have:

$$\begin{array}{llll}
\textbf{Double negation law:} & \neg(\neg A) & \equiv & A \\
\textbf{Commutative laws:} & A \vee B & \equiv & B \vee A \\
 & A \wedge B & \equiv & B \wedge A \\
\textbf{Associative laws:} & ((A \vee B) \vee C) & \equiv & (A \vee (B \vee C)) \\
 & ((A \wedge B) \wedge C) & \equiv & (A \wedge (B \wedge C))
\end{array}$$

$$\begin{array}{lrcl} \textbf{Distributive laws:} & (A \vee (B \wedge C)) & \equiv & ((A \vee B) \wedge (A \vee C)) \\ & (A \wedge (B \vee C)) & \equiv & ((A \wedge B) \vee (A \wedge C)) \\ \textbf{De Morgan's laws:} & \neg(A \vee B) & \equiv & \neg A \wedge \neg B \\ & \neg(A \wedge B) & \equiv & \neg A \vee \neg B \end{array}$$

For instance, the logical equivalence of formulas in the De Morgan's laws can be verified by the following truth tables (*see Table 2.2*).

| $A$ | $B$ | $\neg(A \vee B)$ | $\neg A \wedge \neg B$ | $\neg(A \wedge B)$ | $\neg A \vee \neg B$ |
|---|---|---|---|---|---|
| *true* | *true* | *false* | *false* | *false* | *false* |
| *true* | *false* | *false* | *false* | *true* | *true* |
| *false* | *true* | *false* | *false* | *true* | *true* |
| *false* | *false* | *true* | *true* | *true* | *true* |

Table 2.2: Truth tables validating de Morgan's laws.

In many situation it is convenient to exclude from formulas implication and equivalence connectives. This can be achieved by the means of the following equivalence laws expressing formulas with implication and equivalence by logically equivalent formulas employing only negation, conjunction and disjunction.

$$A \Rightarrow B \equiv \neg A \vee B$$

$$A \Leftrightarrow B \equiv (\neg A \vee B) \wedge (A \vee \neg B)$$

The above can be verified by constructing the appropriate truth tables (*see Table 2.3*).

Given formulas $F_1, F_2, \ldots, F_n$ and a formula $G$, $G$ is said to be a *logical consequence* of $F_1, F_2, \ldots, F_n$ if for any interpretation under which $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ is true, $G$ is also true. If that is the case, $F_1, F_2, \ldots F_n$ are called *postulates*, *premises* or *axioms* of $G$. If the postulates are valid formulas then their logical consequence must be valid also. Clearly, a tautology is a

| $A$ | $B$ | $\neg(A \Rightarrow B)$ | $\neg A \vee B$ | $A \Leftrightarrow B$ | $(\neg A \vee B) \wedge (A \vee \neg B)$ |
|---|---|---|---|---|---|
| *true* | *true* | *true* | *true* | *true* | *true* |
| *true* | *false* | *false* | *false* | *false* | *false* |
| *false* | *true* | *true* | *true* | *false* | *false* |
| *false* | *false* | *true* | *true* | *true* | *true* |

Table 2.3: Meaning of implication and equivalence connectives.

formula which is a logical consequence of an empty set of formulas. We will denote a logical consequence of $G$ from $F_1, F_2, \ldots, F_n$ as $F_1, F_2, \ldots, F_n \models G$.

**Theorem 1 (Deduction theorem)** *Given formulas $F_1, F_2, \ldots, F_n$ and a formula $G$, $G$ is a logical consequence of $F_1, F_2, \ldots, F_n$ if and only if $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ is a valid formula.*

*Proof.* Suppose $F_1, F_2, \ldots, F_n \models G$, i.e.: $G$ is a logical consequence of $F_1, F_2, \ldots, F_n$. Let $I$ be some interpretation. $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ can either be true or false in that interpretation. If it is true, $G$ must also be true (as it is assumed to be a logical consequence) and hence, by the truth table of the implication connective $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ is true. If $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ is false, by the truth table $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ must still be true whatever the logical value of $G$ is.

Conversely, assume $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ to be valid. But, by the truth table, for it to be valid when $G$ is true, $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ must also be true. □

The deduction theorem allows us to formulate the following easy corollary:

**Corollary 1 (Contradiction corollary).** *Given formulas $F_1, F_2, \ldots, F_n$ and a formula $G$, $G$ is a logical consequence of $F_1, F_2, \ldots F_n$ if and only if $F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge \neg G$ is unsatisfiable.*

*Proof.* By the deduction theorem, $(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$ is a valid formula when $G$ is a logical consequence of $F_1, F_2, \ldots, F_n$. Hence, $\neg((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G)$ should be inconsistent. But, based on the equivalence laws established for the propositional calculus $\neg((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G)$ is equivalent to $\neg(\neg(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \vee G)$ which is equivalent to $(F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge \neg G)$. □

Deduction theorem and contradiction corollary demonstrate an approach to mechanical theorem proving. They show that proving a fact that some formula is a logical consequence of a finite set of formulas is equivalent to showing validity or unsatisfiability in propositional calculus of another related formula. This, of course, enables to formalize an axiom system for some particular domain as a set of formulas $A_1, A_2, \ldots, A_n$ and further prove a theorem $T$ in that domain by showing validity in propositional calculus of $A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow T$ or unsatisfiability of $A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg T$.

### 2.1.2 Deductive treatment of propositional calculus

Beside the purely semantic treatment presented in the previous section, we can also build a deductive system for propositional logic where we will show if a formula is valid by demonstrating that it can be derived from the logic axioms by applying the inference rules.

Although the propositional logic surrenders completely to the method of truth tables which allows us to establish validity and inconsistency of formulas and hence by the deduction theorem also whether a formula is logically implied by other formulas, it may, however, be relatively costly. Indeed, we need to evaluate $2^n$ interpretations for a formula of $n$ atoms. For some formulas deductive strategy may, perhaps, present a less costly alternative.

Thus, alternatively to the semantic definition for a valid formula, we may give a deductive one.

**Definition 2** *For any formulas $A$, $B$ and $C$ a* **valid** *formula is:*

- *One of the axioms:*

  **(A1)** $(A \Rightarrow (B \Rightarrow A))$

  **(A2)** $((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$

  **(A3)** $((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))$

- *A conclusion of* modus ponens *inference rule:*

$$\frac{A, (A \Rightarrow B)}{B}$$

  *which states that formula $B$ is valid if formulas $A$ and $(A \Rightarrow B)$ were valid.*

It should be noted that the above axioms do not belong to the language of propositional calculus. They are implicitly quantifying over all subformulas $A$, $B$ and $C$ and thus every axiom actually describes an infinite number of valid formulas.

The recursive definition above, known as Frege-Lukasiewicz deductive system [Bu98], enumerates all formulas which are valid under propositional calculus. By showing that the axioms are true under all interpretations and by demonstrating that modus ponens inference rule preserves validity it is possible to show that every deductively derived theorem is a tautology. We can also demonstrate that every tautology can be deductively derived and hence semantic and deductive definitions can be shown to be equivalent.

The axiomatization presented above uses only negation and implication connectives. We can further define other connectives as follows: $(A \wedge B)$ to mean $\neg(A \Rightarrow \neg B)$, $(A \vee B)$ to mean $\neg A \Rightarrow B$ and $(A \Leftrightarrow B)$ to mean $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

**Example 2** *Show validity of $F \Rightarrow F$.*

*From* **A1** *where $A$ is $F$ and $B$ is $(F \Rightarrow F)$ and from* **A2** *where $A$ is $F$,* *B is $(F \Rightarrow F)$ and C is F by modus ponens obtain:*

$$\frac{(F \Rightarrow ((F \Rightarrow F) \Rightarrow F)), \quad \begin{array}{c}((F \Rightarrow ((F \Rightarrow F) \Rightarrow F)) \Rightarrow \\ ((F \Rightarrow (F \Rightarrow F)) \Rightarrow (F \Rightarrow F)))\end{array}}{((F \Rightarrow (F \Rightarrow F)) \Rightarrow (F \Rightarrow F))}$$

*From the obtained formula and from* **A1** *where A is F and B is F by modus ponens obtain:*

$$\frac{(F \Rightarrow (F \Rightarrow F)) \Rightarrow (F \Rightarrow F), (F \Rightarrow (F \Rightarrow F))}{F \Rightarrow F}$$

*which is the sought formula now proven to be valid.*

Other axiomatizations based on different sets of connectives are also possible (e.g.: Hilbert and Ackerman axiomatization based on negation and disjunction [Me64]). However, using deductive systems of this type (known as Hilbert-type) on practice is rather inconvenient, since the proofs become quite long even for simple theorems. Most practical implementations opt for either Gentzen-type deductive systems (*see Section 2.2*) or resolution based deductive systems (*see Section 3.2*) which are more attractive to employ in an automated theorem prover [Le97].

## 2.2 First-order predicate calculus

Propositional calculus is not expressive enough to describe many axiom systems. It is often necessary to consider the internal structure of the propositions which is not possible in the propositional logic. For this purpose, in the predicate calculus we use atoms built from *predicates* of *functions* and *quantified variables* instead of simple propositions. The functions and the quantified variables express some objects whereas the predicates express objects' qualities. Thus, the predicate calculus is convenient to obtain formalizations of common axiom systems of mathematics which usually operate with some set of objects.

The set $\mathcal{U}$ of all objects is called the *universe*. The functions and the predicates operate in the universe.

If $\mathcal{A}$ and $\mathcal{B}$ are sets, a *mapping* from $\mathcal{A}$ to $\mathcal{B}$ is an assignment of an object in $\mathcal{B}$ to each object in $\mathcal{A}$. A mapping from a set of n-tuples of objects in a set $\mathcal{A}$ into a set $\mathcal{B}$ is called an n-ary *function* from $\mathcal{A}$ to $\mathcal{B}$. A subset of n-tuples in $\mathcal{A}$ is called an n-ary *predicate* in $\mathcal{A}$.

Semantically, we can consider a predicate as a mapping of n-tuples in a set $\mathcal{A}$ into the set of truth values $\{true, false\}$. Thus, if an n-tuple belongs to the subset defined by the predicate, we describe such a situation with the mapping into the truth value of *true*, otherwise the result is described by the truth value of $false$.

Members of the universe of an axiom system are referred to as *individuals*. The functions from the universe to the universe are called *individual functions* and predicates in the universe - *individual predicates*. A 0-ary individual function always maps into the same individual. Thus, we call 0-ary functions as *constants*. Constants are used to refer to particular individuals. A binary predicate of *equality* is of special importance for many axiom systems and thus often occupies a special place among predicates.

With the logical connectives employed in propositional calculus, we can express complex facts about multiple propositions. We cannot, however, express even a simple general law which is true for multiple objects. For that purpose we introduce into the language of predicate calculus additional logical symbols describing *individual variables* and their *quantifiers*. Two quantifiers are used: the *universal quantifier* "$\forall$" and the *existential quantifier* "$\exists$". The former allows us to express formulas which are true for any individual $x$ and the latter formulas which are true for some individual $x$. A quantifier of a variable precede a formula with that variable occurring (e.g.: $(\forall x)F(x)$). The formula immediately following the quantifier is in the quantifiers scope. An occurrence of a variable is called *bounded* if it is in a scope of a quantifier of that variable. Other occurrences of variables are called *free*.

**Definition 3** *A* **term** *is:*

- *A variable.*
- *If $t_1, \ldots, t_n$ are terms, and $f$ an n-ary function, then $f(t_1, \ldots, t_n)$ is a term. (Since the constants are 0-ary functions, any constant $g()$ is also a term).*

A term that does not contain any variables is called a *ground term*.

**Definition 4** *A* **formula** *of predicate calculus is:*

- *If $t_1, \ldots, t_n$ are terms and $P$ an n-ary predicate, then $P(t_1, \ldots, t_n)$ is an atomic formula.*
- *If $A$ is a formula, then so is $(\neg A)$.*
- *If $A$ and $B$ are formulas, then so are $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$.*
- *If $A[x]$ is a formula and $x$ is a free variable occurring in that formula then $(\forall x)A[x]$ and $(\exists x)A[x]$ are formulas.*

The *language* of first-order predicate calculus is defined to be a language in which the formulas are of the syntax as described above. The name – *first-order* refers to what is allowed to be quantified in the language. It is possible to define other languages, called *higher-order* languages, where, for instance, predicates and functions are allowed to be quantified.

It should be noted that formula definition given above also enumerates formulas with free variables remaining. Since it is hard to assign any reasonable meaning to such formulas we will assume that all interpretations treat free variables as if they are universally quantified by default.

We will use the same convention of logic connectives precedence and the same definition of semantic meaning of the connectives as established in Section 2.1.

### 2.2.1 Semantics of predicate calculus

An *interpretation* of a formula $F$ in the first-order predicate calculus consists of a non-empty subset of the universe individuals $\mathcal{D}$ (called the *domain of the interpretation*), and an assignment of values to each function symbol and predicate in the following way: Each n-ary function symbol is assigned a mapping from n-tuples in $\mathcal{D}$ into $\mathcal{D}$ (hence each constant is assigned a single element in $\mathcal{D}$). Each n-ary predicate symbol is assigned a subset of n-tuples in $\mathcal{D}$.

The concepts of validity, satisfiability and contradiction are the same as that in the propositional logic.

**Example 3** *Somewhat informally, the formula* $(\forall x)(P(x))$ *in the interpretation over domain* $\mathcal{D} = \{a(), b(), c()\}$ *where* $P = \{(a()), (c())\}$ *is false, since* $P(b())$ *is false (it is not in the subset describing the predicate* $P$*), and thus* $(\forall x)(P(x))$ *is not true for all* $x \in \mathcal{D}$.

•

The equivalence laws demonstrating pairs of equivalent propositional formulas are also true in the predicate logic and they can be augmented by the equivalence laws for the formulas having quantifiers. We introduce scoping rules for quantifiers in the following way:

$$(Qx)(F[x] \vee G) \equiv (Qx)(F[x]) \vee G$$

$$(Qx)F[x] \wedge G \equiv (Qx)(F[x]) \wedge G$$

where $Q$ is any quantifier, $F[x]$ is a term containing quantified variable $x$, and $G$ is a term which does not depend on $x$. Clearly, a term which does not depend on a variable can be brought out of the scope of that variable's quantifier without compromising validity properties. In general we can bring the quantifiers in front of the terms by proper variable renaming. Thus assuming that new, distinct variable $y$ is introduced:

$$(Q_1x)F[x] \vee (Q_2x)H[x] \equiv (Q_1x)(Q_2y)(F[x] \vee H[y])$$

$$(Q_1x)F[x] \wedge (Q_2x)H[x] \equiv (Q_1x)(Q_2y)(F[x] \wedge H[y])$$

We additionally introduce laws for negation of quantifiers:

$$\neg((\forall x)F[x]) \equiv (\exists x)(\neg F[x])$$

$$\neg((\exists x)F[x]) \equiv (\forall x)(\neg F[x])$$

These laws can be easily proven. Let $I$ be some interpretation over some domain $\mathcal{D}$. If $\neg((\forall x)F[x])$ is true in $I$, then $(\forall x)F[x]$ is false in $I$, which means that there exists at least one element of $\mathcal{D}$ for which $F[x]$ is false. Therefore $(\exists x)(\neg F[x])$ is true. The other law can be proven in a similar way.

The deduction and contradiction theorems which were true for the propositional calculus can also be shown to be true for the first-order predicate calculus. However, whereas for propositional logic any formula had only a finite number of interpretations ($2^n$ where $n$ is the number of distinct atoms in that formula), in predicate logic there is a potentially infinite number of interpretations for formulas due to the fact that the universe is infinite and thus there may be an infinite number of interpretation domains.

### 2.2.2 Deductive treatment of predicate calculus

Besides a purely semantic treatment, similarly to the propositional calculus, it is possible to give a deductive system for predicate calculus. Also as was the case with propositional logic, many different axiomatizations are possible most of which fall into two categories of either Gentzen or Hilbert style. The former systems are given for formulas in *sequent* notation and usually contain few axioms but multiple inference rules [SJ97]. The latter systems have multiple axioms yet relatively few inference rules [Me64].

Let's consider a Gentzen-type axiomatization. These are often known as sequent calculi since they operate on sentences essentially expressing logical implication of a set of formulas from another set of formulas. Thus a *sequent* is a sentence of the type

$$\Gamma \models \Delta$$

where $\Gamma$ and $\Delta$ are finite sets of formulas: $G_1, \ldots, G_n$ and $D_1, \ldots, D_m$ respectively. Semantically, a sequent $\Gamma \models \Delta$ holds if every interpretation $I$ which makes all formulas of $\Gamma$ true, also makes at least one formula from $\Delta$ also true. The equivalent deductive definition of the valid sequent is as follows:

**Definition 5** *For any formulas $A$, $B$, and sets of formulas $\Gamma$ and $\Delta$, a* **valid** *sequent is:*

- *The axiom: $A \models A$*
- *Assuming that the hypotheses of the inference rules are valid sequents the conclusions are valid sequents:*

$$\frac{\Gamma, A, B \models \Delta}{\Gamma, A \wedge B \models \Delta} \qquad \frac{(\Gamma \models \Delta, A), (\Gamma \models \Delta, B)}{\Gamma \models \Delta, A \wedge B}$$

$$\frac{(\Gamma, A \models \Delta), (\Gamma, B \models \Delta)}{\Gamma, A \vee B \models \Delta} \qquad \frac{\Gamma \models \Delta, A, B}{\Gamma \models \Delta, A \vee B}$$

$$\frac{(\Gamma \models \Delta, A), (\Gamma, B \models \Delta)}{\Gamma, A \Rightarrow B \models \Delta} \qquad \frac{\Gamma, A \models \Delta, B}{\Gamma \models \Delta, A \Rightarrow B}$$

$$\frac{\Gamma \models \Delta, A}{\Gamma, \neg A \models \Delta} \qquad \frac{\Gamma, A \models \Delta}{\Gamma \models \Delta, \neg A}$$

$$\frac{\Gamma, A\{x|t\}, (\forall x)A \models \Delta}{\Gamma, (\forall x)A \models \Delta} \qquad \frac{\Gamma \models \Delta, A[x]}{\Gamma \models \Delta, (\forall x)A[x]}$$

$$\frac{\Gamma, A[x] \models \Delta}{\Gamma, (\exists x)A \models \Delta} \qquad \frac{\Gamma \models \Delta, A\{x|t\}, (\exists x)A[x]}{\Gamma \models \Delta, (\exists x)A[x]}$$

Sequent calculi allow expressing an axiomatization for some particular domain as a set of sequents and to further build proofs using the inference rules. The proofs found in a sequent calculus are usually quite intuitive for human readers, however, mechanical procedures to find proofs are both complex in implementation and relatively inefficient.

Whereas using sequent calculi on practice in a mechanical procedure is problematic, Hilbert-type deductive systems in unmodified form are hardly usable at all. These remain to be theoretical devices used to obtain a more practical types of deductive systems.

The following Hilbert-type deductive system uses five axioms and two inference rules. It can be shown that both deductive systems recursively enumerate all tautologies of the first-order predicate calculus and are thus equivalent.

**Definition 6** *For any formulas $A$, $B$ and $C$, a* **valid** *formula is:*

- *One of the axioms:*

$$(A \Rightarrow (B \Rightarrow A))$$

$$((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$$

$$((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))$$

$$(\forall x)A[x] \Rightarrow A[t]$$

  *where $x$ is a variable, $A[x]$ a formula containing $x$ and $t$ some term.*

$$(\forall x)(A \Rightarrow B[x]) \Rightarrow (A \Rightarrow (\forall x)B[x])$$

  *where $B[x]$ is a formula containing a variable $x$ and $A$ a formula where $x$ does not occur.*

- *A conclusion of* modus ponens *inference rule:*

$$\frac{A, (A \Rightarrow B)}{B}$$

  *which states that formula $B$ is valid if formulas $A$ and $(A \Rightarrow B)$ were valid.*

- *A conclusion of* generalization *inference rule:*

$$\frac{A}{(\forall x)A}$$

  *which states that formula $(\forall x)A$ is valid if formula $A$ was valid.*

It should be underlined again that, as was the case with axiomatization for propositional calculus, the above axioms (both in Gentzen and Hilbert-type systems) do not belong to the language of first-order predicate calculus. They are implicitly quantifying over all subformulas $A$, $B$ and $C$ and thus every axiom actually describes an infinite number of valid formulas.

With the Hilbert-type deductive system described, we can build an axiom system for some particular domain as a set of formulas $A_1, A_2, \ldots, A_n$ and further prove a theorem $T$ in that domain by showing validity in first-order predicate calculus of a formula $A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow T$ or unsatisfiability of a formula $A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg T$.

Many such axiomatizations require the notion of *equality* of individuals. This can be achieved by introducing an equality predicate: "Equal". [2] The behavior of equality can be described by the following five axioms:

$$(\forall x)(Equal(x,x))$$

$$(\forall x)(\forall y)(Equal(x,y) \Rightarrow Equal(y,x))$$

$$(\forall x)(\forall y)(\forall z)(Equal(x,y) \wedge Equal(y,z) \Rightarrow Equal(x,z))$$

$$(\forall P)(\forall x_1 \ldots x_n, y_1 \ldots y_n)(Equal(x_1,y_1) \wedge \ldots \wedge Equal(x_n,y_n)) \Rightarrow (P(x_1,\ldots,x_n) \Rightarrow P(y_1,\ldots,y_n))$$

$$(\forall f)(\forall x_1 \ldots x_n, y_1 \ldots y_n)(Equal(x_1,y_1) \wedge \ldots \wedge Equal(x_n,y_n)) \Rightarrow Equal(f(x_1,\ldots,x_n), f(y_1,\ldots,y_n))$$

The first-order predicate calculus with equality predicate defined[3] is known as *first-order predicate calculus with equality.*

An axiom system formulated in the language of first-order predicate calculus for some particular domain is called a *first-order* theory.

A first-order theory is said to be *inconsistent* if every formula of the theory is a theorem. Otherwise the theory is called *consistent.* Clearly,

[2]We will denote the equality predicate in the prefix notation $Equal(x,y)$ used for all other predicates, as opposed to infix notation ($x = y$) which is at times used in the literature.

[3]Equality substitutivity axioms represented here are of higher order since they quantify over all individual predicates and function. What it implies is that every predicate and function needs its own first-order substitutivity axiom

from an inconsistent theory we can deduce both a fact $A$ and its negation $\neg A$ being both valid.

A theory is called *decidable* if there exists a procedure (an algorithm) to test whether or not formulas are true in that theory. A theory is called *complete* if it is consistent and decidable.

As shown by Church, first-order predicate calculus itself is undecidable, that is, there is no general decision procedure to test validity of formulas in first-order predicate calculus. However there are decision procedures to demonstrate validity of formulas which are indeed valid. Such procedures, however, may not terminate for invalid formulas.

Only in recent years the attention turned to the complexity of decision problems. Although it appears that once a problem is shown to be decidable using a decision procedure on practice is a trivial matter, many decidable decision problems appear to be intractable in a sense that they require exponential (or worse) number of execution steps. For instance, although testing satisfiability of formulas of propositional logic is a decidable problem it appears that any algorithm to compute satisfying interpretations needs exponential number of execution steps in the worst case. Although this result has not been proven yet, the strong conjecture is that the class of problems solvable in non-deterministic polynomial time ($NP$) (to which propositional satisfiability belongs) is distinct from the class of tractable problems solvable in polynomial time ($P$).

### 2.2.3 Normal forms of formulas of predicate calculus

The deductive systems described before suffer from complexity. This makes them fairly inefficient for use on practice in a mechanical theorem prover. The complexity comes from the richness of the underlining language. If we can somehow restrict this richness and allow only formulas of a particular form (assuming that any formula can be transformed into such form), perhaps a simpler deductive system can be build. This is indeed the case.

The Robinson's resolution (*see Section 3.2*) is such a system operating on formulas in *clause form* which is closely related to *conjunctive normal form.*

The formula in *conjunctive normal form* is such that it is a chain of conjuncted clauses which are chains of disjuncted literals, such as $((A \vee B \vee \neg C) \wedge (A \vee C) \wedge (D \vee \neg A))$ where each literal is either an atomic formula or its negation.

To transfer a formula into conjunctive normal form we use the following procedure which involves multiple invocations of the equivalence laws:

- Use the definitions of implication and equivalence to obtain formulas having only the connectives of negation, conjunction and disjunction.
- Use De Morgan's laws and the double negation law to bring the negations immediately before the atoms.
- Repeatedly use the distributive laws to distribute conjunctions over disjunctions.

Since the procedure uses equivalence laws only, the validity properties of the resulting formulas are the same as that of the argument formulas.

**Example 4** *Transform* $\neg(A \wedge (B \Rightarrow C))$ *into conjunctive normal form.*
*Firstly, eliminate implication and equivalence connectives:*

$$\neg(A \wedge (B \Rightarrow C)) \equiv \neg(A \wedge (\neg B \vee C))$$

*Bring the negation before the atoms:*

$$\neg(A \wedge (\neg B \vee C)) \equiv \neg A \vee \neg(\neg B \vee C)$$

$$\neg A \vee \neg(\neg B \vee C) \equiv \neg A \vee (B \wedge \neg C)$$

*Distribute conjunctions over disjunctions:*

$$\neg A \vee (B \wedge \neg C) \equiv (\neg A \vee B) \wedge (\neg A \vee \neg C)$$

This procedure works for both propositional and predicate logic. However, presence of quantifiers in the predicate logic requires special treatment. A formula of first-order predicate calculus is said to be in *prenex normal form* if all quantifiers precede the reminder of the formula containing the terms. The preceding quantifiers are called the *prefix* of the formula whereas the reminder of the formula is called the *matrix*.

The following procedure which uses the equivalence laws for quantifiers can be employed to obtain formulas in the prenex normal form:

- Use the definitions of implication and equivalence to obtain formulas having only the functions of negation, conjunction and disjunction.
- Use De Morgan's laws, the double negation law, and the laws for quantifier negation to bring the negations immediately before the atoms.
- Use the quantifier scoping laws and variables renaming to bring the quantifiers in front of the matrix.

A prenex form can be combined with conjunctive normal form so that a formula is first transformed into the prenex form and further the matrix is transformed into the conjunctive normal form.

Most proof procedures operate on formulas in even simpler form, referred to as the *standard* or *Skolem normal form* which represents quantifiers implicitly rather then explicitly.

To transfer a formula into the standard form the following procedure is used:

- Transfer a formula into prenex form.
- Transfer the matrix of the resulting formula into conjunctive normal form.
- Eliminate existential quantifiers by substituting existentially quantified variables for Skolem functions or constants in the following way:

Let $(Q_r x_r)$ be the leftmost existential quantifier in the prefix of a formula: $(Q_1 x_1)(Q_2 x_2) \dots (Q_n x_n)$ we replace all occurrences of $x_r$ in the matrix by a new function symbol of $r-1$ variables: $f(x_1, \dots, x_{r-1})$ and remove $(Q_r x_r)$ from the prefix.

- Eliminate universal quantifiers by assuming that all remaining variables are quantified universally.

A formula transferred into the standard normal form can be represented as a set of *clauses*, where each clause is a disjunction of literals.

**Theorem 2 (Standard form completeness)** *Let $F'$ be a a formula in standard form derived from formula $F$. $F$ is inconsistent if and only if $F'$ is inconsistent.*

*Proof.* Without loss of generality, let's assume that a formula $F$ is already in prenex form. Let $Q_r$ be the first existential quantifier. Thus a formula $F = (Q_1 x_1) \dots (Q_r x_r) \dots (Q_n x_n) M[x_1, \dots, x_n]$ is transformed into

$$F' = (\forall x_1) \dots (\forall x_{r-1})(Q_{r+1} x_{r+1}) \dots (Q_n x_n) M[x_1 \dots f(x_1, \dots, x_{r-1}) \dots x_n]$$

where $f$ is a Skolem function. We must show that $F$ is inconsistent if and only if $F'$ is inconsistent.

Suppose $F$ is inconsistent. If $F'$ is consistent there is an interpretation $I$ such that $F'$ is true in $I$. That would imply that for all $x_1, \dots, x_{r-1}$ there is an element (for instance $f(x_1, \dots, x_{r-1})$) such that

$$(Q_{r+1} x_{r+1}) \dots (Q_n x_n) M[x_1, \dots, f(x_1, \dots, x_{r-1}), \dots, x_n]$$

is true. Thus $F$ must be true and hence $F'$ is inconsistent if so is $F$.

Suppose $F'$ is inconsistent. If $F$ is consistent there is an interpretation $I$ such that $F$ is true in $I$. That would imply that for all $x_1, \dots, x_{r-1}$ there exists an $x_r$ such that $(Q_{r+1} x_{r+1}) \dots (Q_n x_n) M[x_1, \dots, x_r, \dots, x_n]$ is true. We extend the interpretation $I$ to include a mapping $f$ from $x_1, \dots, x_{r-1}$

to $x_r$ and denote the extended interpretation $I'$. However that would mean that $F'$ must be true in $I'$. Thus $F$ must be inconsistent if so is $F'$. $\square$

The standard form completeness theorem demonstrates that we can use standard forms of formulas as opposed to formulas themselves in proof procedures. Since formulas in standard form have a much simpler structure, this can be exploited to obtain a simpler and more efficient algorithms.

# Chapter 3

# Fundamental proving procedures

The mechanical procedures to prove theorems for particular domains rely on the techniques allowing to show validity or unsatisfiability of other related formulas in a logic calculus. Many of the procedures show unsatisfiability for sets of clauses and thus before such methods are applied, the original formulas representing the axioms and the theorem must be normalized. Although the approaches involving normal forms are quite efficient, another direction of automated reasoning is that of pursuing natural deduction on essentially unnormalized formulas. These approaches are often based on Gentzen-type axiom systems (*see Section 2.2.2*) and operate with larger sets of rules of inference. One often cited advantage of that approach is that the derived proofs are more meaningful for human readers. However the efficiency of natural deduction is often inferior to the procedures operating with normalized formulas, which remain the mainstream direction for practical automated theorem proving. Most of these procedures implicitly or explicitly rely upon Herbrand's theorem which allows us to express the semantic meaning of formulas by convenient enumeration of its appropriate (and not all possible) interpretations.

## 3.1 Herbrand's theorem

A formula $F$ is valid if and only if the negation of that formula $\neg F$ is unsatisfiable. Any formula can be expressed by a logically equivalent set of clauses $\mathcal{S}$. A set $\mathcal{S}$ of clauses is unsatisfiable if and only if it is false under all interpretations over all domains. However, checking mechanically this fact over all domains is impossible since there is an infinite number of such domains. Alternatively, we can attempt to find a single special domain $\mathcal{H}$ such that $\mathcal{S}$ is unsatisfiable if and only if it is false under all interpretations over this particular domain. Such a domain indeed exists and it is known as *Herbrand universe.*

Let $\mathcal{H}_0$ be the set of all constants appearing in $\mathcal{S}$. (If there is no constants in $\mathcal{S}$ we choose $\mathcal{H}_0 = \{a()\}$). For $i = 0, 1, \ldots$ Let $\mathcal{H}_i$ be the union of $\mathcal{H}_{i-1}$ and the set of all terms of the form $f_n(t_1, \ldots, t_n)$ where $t_i \in \mathcal{H}_i$ for $i = 0, \ldots, n$ and $f_n$ is any n-ary function occurring in $\mathcal{S}$. Each $\mathcal{H}_i$ is called *i-level constant set* of $\mathcal{S}$ and $\mathcal{H}_\infty$ is called the *Herbrand universe* of $\mathcal{S}$.

We can, thus, define interpretations over the domain $\mathcal{H}_\infty$, called *$\mathcal{H}$-interpretations* as follows: Given a set of clauses $\mathcal{S}$ An interpretation $I^\star$ is said to be an $\mathcal{H}$-interpretation of $\mathcal{S}$ if it satisfies the following: $I$ maps constants to themselves. If $f$ is an n-ary function and $h_1, \ldots, h_n$ are elements of $\mathcal{H}_\infty$ in $I$, $f$ assigns a function mapping $(h_1, \ldots, h_n)$ (an n-tuple in $\mathcal{H}_\infty$) into $f(h_1, \ldots, h_n)$, an element in $\mathcal{H}_\infty$.

Any interpretation $I$ over some domain $\mathcal{D}$, should have a corresponding $\mathcal{H}$-interpretation $I^\star$ (thus over the Herbrand universe) which satisfies the following. If $h_1, \ldots, h_n$ are elements of $\mathcal{H}$, every $h_i$ can be mapped to some $d_i$ in $\mathcal{D}$. The same truth value assigned to $P(d_1, \ldots, d_n)$ in $I$ should be assigned to $P(h_1, \ldots, h_n)$ in $I^\star$. If an interpretation $I$ over some domain $\mathcal{D}$ satisfies a set $\mathcal{S}$ then any of the $\mathcal{H}$-interpretations $I^\star$ corresponding to $I$ also satisfy $\mathcal{S}$.

**Theorem 3 (Herbrand unsatisfiability)** *A set $\mathcal{S}$ of clauses is unsatisfiable if and only if $\mathcal{S}$ is false under all $\mathcal{H}$-interpretations of $\mathcal{S}$.*

*Proof.* Clearly, if $\mathcal{S}$ is unsatisfiable it must be false under all interpretations which includes all $\mathcal{H}$-interpretations.

Assume $\mathcal{S}$ is false under all $\mathcal{H}$-interpretations. Suppose $\mathcal{S}$ is satisfiable. Then there is an interpretation $I$ over some $\mathcal{D}$ such that $\mathcal{S}$ is true. But we can construct $I^*$ – an $\mathcal{H}$-interpretation corresponding to $I$ which by the assumption was false. Since $\mathcal{S}$ should have the same truth value under $I^*$ and $I$, $\mathcal{S}$ must be unsatisfiable. □

Herbrand unsatisfiability theorem demonstrates that a task of finding the unsatisfiability of a set of clauses can be done by checking only the interpretations over the Herbrand universe of that set of clauses. Thus the task of checking unsatisfiability of some formula in first-order predicate calculus is reduced to finding the clause set of its standard form and essentially propositional task of listing $\mathcal{H}$-interpretations.

### 3.1.1 Semantic trees

Even though we only need to consider the $\mathcal{H}$-interpretatons to check unsatisfiability, it is also true that Herbrand universe of a formula containing functions other then constants is infinite and thus there may be an infinite number of $\mathcal{H}$-interpretations which are infinite in length. However, Herbrand universe has a particular structure which can be exploited using the notion of a *semantic tree* which allows us to express unsatisfiability of a set $\mathcal{S}$ in a finite manner. Semantic trees where proposed by Robinson in 1968 and refined by Kowalski and Hayes in 1969.

Given a set $\mathcal{S}$ of clauses, let $\mathcal{A}$ be the atom set of $\mathcal{S}$. A semantic tree for $\mathcal{S}$ is a tree (*see Figure 3.1*), where every edge is labelled with a finite set of atoms or atom negations from $\mathcal{A}$ in such a way that:

- For each node $N$, there are finitely many immediate edges $E_1, \ldots, E_n$ from $N$, and the disjunction of all atoms attached to $E_1, \ldots, E_n$ is a tautology.

- For each node $N$, let $I(N)$ (called the ***partial interpretation***) be the union of all the sets of atoms attached to the edges connecting the root of the tree with the node $N$. $I(N)$ may not contain a complementary pair (an atom and its negation).

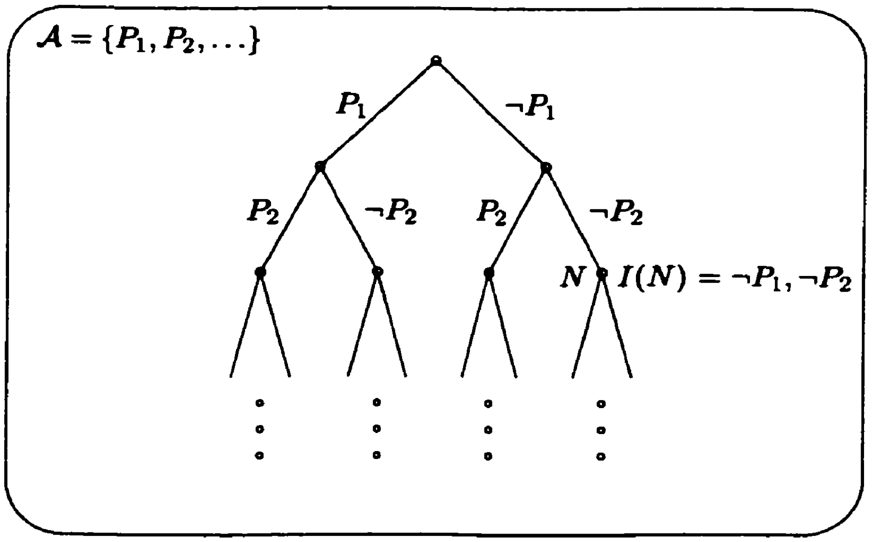
$\mathcal{A} = \{P_1, P_2, \ldots\}$
$P_1$
$\neg P_1$
$P_2$
$\neg P_2$
$P_2$
$\neg P_2$
$N$
$I(N) = \neg P_1, \neg P_2$

Figure 3.1: A semantic tree for an atom set $\mathcal{A}$.

A semantic tree in which every path from the root node down the tree contains every atom or a negated atom of the set $\mathcal{A}$ is referred to as a complete semantic tree. It can be seen that a complete semantic tree of a Herbrand universe of a clause set $S$ corresponds to exhaustive survey of all possible $\mathcal{H}$-interpretations. Since the Herbrand universe may well be an infinite set with the corresponding infinite complete semantic tree the following notion is crucial: A node $N$ is called a ***failure node*** if $I(N)$ falsifies some ground instance of a clause in $S$ and there is no other failure nodes on the path from $N$ to the root of the tree.

A semantic tree is closed if and only if its every branch terminates in a failure node.

**Theorem 4 (Herbrand's theorem)** *A set of clauses is unsatisfiable if and only if its every complete semantic tree is closed.*

*Proof.* Suppose $S$ in unsatisfiable. Let $T$ be a complete semantic tree for $S$. Every branch of $\mathcal{T}$ represents an interpretation which must be false if $S$ is unsatisfiable, but then every branch will terminate in a failure node and $T$ will be closed. Conversely if every $\mathcal{H}$-interpretation of $S$ is false, $S$ is unsatisfiable. □

Herbrand's theorem thus suggests a procedure to check unsatisfiability of formulas in first-order predicate calculus: attempt to build a closed semantic tree for a clause set. This theorem is also important in showing completeness of other proof procedures such as that of resolution principle.

### 3.1.2 The method of Davis and Putnam

Implementations of early mechanical proof procedures based on Herbrand's theorem such as the one done by Gilmore tended to be inefficient even for proving simple theorems. In 1960 Davis and Putnam introduced a procedure for testing unsatisfiability of a set of ground clauses whose efficiency was considerably better compared with implementations of earlier procedures [Lo78].

Davis and Putnam's procedure for testing unsatisfiability of a set of clauses $S$ can be expressed as the following four rules:

- **The Tautology Rule**: Remove all clauses which are tautologies.
- **One Literal Rule**: If there is a unit ground clause $L$ in $S$, obtain $S'$ from $S$ by deleting those ground clauses in $S$ which contain $L$. If $S'$ is empty, $S$ is satisfiable. If $S'$ is not empty, obtain a set $S''$ from $S'$ by deleting occurrences of $\neg L$ in all clauses.
- **Pure-Literal Rule**: A literal $L$ is said to be pure if $\neg L$ doesn't occur anywhere in the set. Obtain $S'$ by deleting clauses containing $L$.

- **Splitting Rule**: If it is possible to represent the set $\mathcal{S}$ in the following form:

$$(A_1 \vee L) \wedge \ldots \wedge (A_m \vee L) \wedge (B_1 \vee \neg L) \wedge \ldots \wedge (B_n \vee \neg L) \wedge R$$

  obtain two sets $\mathcal{S}_1 = A_1 \wedge \ldots \wedge A_m \wedge R$ and $\mathcal{S}_2 = B_1 \wedge \ldots \wedge B_n \wedge R$. $\mathcal{S}$ is unsatisfiable if both $\mathcal{S}_1$ and $\mathcal{S}_2$ are unsatisfiable (That is $\mathcal{S}_1 \vee \mathcal{S}_2$ is unsatisfiable.

**Theorem 5 (Soundness of Davis-Putnam procedure)** *If the original set $\mathcal{S}$ was unsatisfiable, the set resulting after application of any rule will also be unsatisfiable.*

*Proof.* Clearly, the Tautology Rule doesn't violate the satisfiability properties. Since a tautology is satisfied by any interpretation, the original set $\mathcal{S}$ is unsatisfiable if and only if the set resulted after removal of tautologies is unsatisfiable.

For the One Literal Rule, suppose $\mathcal{S}''$ is unsatisfiable. Assume $\mathcal{S}$ to be satisfiable, then there is a model $M$ of $\mathcal{S}$ containing literal $L$. For $\mathcal{S}''$, $M$ must satisfy all the clauses that contained $\neg L$ (since $\neg L$ is falsified in $M$) therefore $M$ should satisfy $\mathcal{S}''$. Conversely, suppose $\mathcal{S}$ is unsatisfiable. If $\mathcal{S}''$ is satisfiable, then there is a model $M''$ of $\mathcal{S}''$. But then the union of $M''$ and $L$ would be a model of $\mathcal{S}$ which contradicts the assumption that $\mathcal{S}$ is unsatisfiable.

For the Pure Literal Rule, suppose $\mathcal{S}'$ is unsatisfiable, then $\mathcal{S}$ must be unsatisfiable since the clauses of $\mathcal{S}'$ form a subset with respect to clauses in $\mathcal{S}$. Conversely, suppose $\mathcal{S}$ is unsatisfiable. If $\mathcal{S}'$ is satisfiable, then there is a model $M$ of $\mathcal{S}'$ which doesn't include $L$ or $\neg L$ but that would mean that the union of $M$ and $L$ is a model of $\mathcal{S}$ which was assumed not to have a model.

For the Splitting Rule. Suppose $\mathcal{S}$ is unsatisfiable. Assume $\mathcal{S}_1$ is satisfiable. Then there is a model $M$ of $\mathcal{S}_1$. But that would imply that the union of $M$ and $\neg L$ can satisfy all clauses of $\mathcal{S}$ which was assumed to be unsatisfiable. Conversely, assume $\mathcal{S}_1 \vee \mathcal{S}_2$ to be unsatisfiable. If $\mathcal{S}$ is satisfiable there should be a model $M$ for $\mathcal{S}$. If this model contains $L$ it should have been satisfied in order to satisfy $\mathcal{S}$. But thus $\mathcal{S}_2$ would be satisfiable. If $M$ contains $\neg L$ it can satisfy $\mathcal{S}_1$. Since $M$ has to contain either $L$ or $\neg L$, $\mathcal{S}_1 \vee \mathcal{S}_2$ should be satisfiable which contradicts the assumption. $\square$

Although Davis-Putnam procedure is quite efficient, it is inherently a propositional method, necessarily requiring a set $\mathcal{S}$ to be a set of ground clauses.

## 3.2 Robinson's resolution principle

The resolution principle was proposed by Robinson in 1965 and constituted a major break-through for practical automated theorem proving [Ro65]. This proof procedure is applied to a set of clauses (not necessarily ground clauses as is the case with Davis-Putnam procedure) to find logically implied resolvents.

### 3.2.1 Resolution principle for propositional calculus

Applied to propositional logic, the resolution principle is essentially an extension of Davis-Putnam's one literal rule and is an inference rule which can be formulated similarly to the Cut rule [1] often added into Gentzen-type deductive systems

$$\frac{(A \Rightarrow B), (B \Rightarrow C)}{A \Rightarrow C}$$

The conclusion of this inference is referred to as a *binary resolvent.*

[1] Cut rule formulated in Gentzen-type deductive systems as

$$\frac{(\Gamma_1, A \models \Delta), (\Gamma_2 \models \Delta_2, A)}{\Gamma_1, \Gamma_2 \models \Delta_1, \Delta_2}$$

is unnecessary there. It can be shown that it is redundant with respect to other inference rules.

**Theorem 6 (Soundness of resolution)** *A binary resolvent $C = C_1 \vee C_2$ is a logical consequence of its hypotheses: $H_1 = L \vee C_1$ and $H_2 = \neg L \vee C_2$.*

*Proof.* Let $H_1$ and $H_2$ be true in some interpretation $I$. Assume $C$ is false in $I$, Either $L$ or $\neg L$ must be true in $I$. Without loss of generality, assume $L$ is false, then $C_1$ must have been true and not a unit clause. But that would imply that $C_1 \vee C_2$ , i.e. $C$ is true in $I$. □

A *refutation* is a proof ending with a contradiction. Clearly, since the resolution is shown to be sound, finding a refutation from a set of clauses $\mathcal{S}$, demonstrates unsatisfiability of $\mathcal{S}$. Combined with the result of Contradiction theorem this suggests a mechanical proof procedure: negating the theorem to be proven and generating resolutions from the obtained clause set until the refutation is obtained.

### 3.2.2 Resolution principle for predicate calculus

Necessity to consider the internal structure of the atoms and presence of quantified variables complicates resolution in first-order predicate calculus.

A *substitution* is a finite set of the form $\theta = \{t_1|v_1, \ldots, t_n|v_n\}$ where every $v_i$ is a variable and every $t_i$ is a term not containing $v_i$. When all $t_i$ do not contain variables, $\theta$ is called a *ground substitution.* A substitution with no elements is called an *empty substitution.*

An expression $L\theta$ obtained from a substitution $\theta$ by replacing in $L$ all occurrences of the variables by terms specified by the substitution $\theta$ is called an *instance*. An instance which does not contain any quantified variables is called a *ground instance*. Clearly, two substitutions $\alpha$ and $\beta$ can be combined by composition $\alpha \circ \beta$. The composition of substitutions is associative (that is: $(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$). The empty substitution $\epsilon$ is both left and right identity (that is: $\alpha = \epsilon \circ \alpha = \alpha \circ \epsilon$).

A substitution $\theta$ is called a *unifier* for a set of literals $\mathcal{L} = \{L_1, \ldots, L_n\}$ if and only if $L_1\theta = L_2\theta = \ldots = L_n\theta$. The set $\{L_1, \ldots, L_n\}$ is then said to be *unifiable.* A unifier $\sigma$ for a set $\mathcal{L}$ is said to be the *most general unifier* if and only if for each unifier $\theta$ of that set there is a substitution $\gamma$ such that $\theta = \sigma \circ \gamma$.

To unify a set of terms:

- Start from the first symbols in the terms.
- If the symbols are the same (either functions or variables), advance to the following symbols. If the symbols are different unification fails.
- If the current symbols are: $v_i$ and $t_j$ - a variable and a term not containing $v_i$ add $\{v_i|t_j\}$ to the unifier. Otherwise, if the term $t_j$ does contain $v_i$, unification fails.
- Rewrite the terms so that all substituted variables are replaced by the respective terms.
- Repeat previous two steps till the last symbol.

If two or more literals of the same sign (either all positive or negated) of a clause $C$ have a most general unifier $\theta$ then $C\theta$ with only one of the unified literals remaining is called a *factor*. If $C\theta$ is a unit clause it is called a *unit factor.*

If $C_1$ and $C_2$ are two clauses so that $L_1$ and $\neg L_2$ are two literals in $C_1$ and $C_2$ respectively and $L_1$ and $\neg L_2$ have a most general unifier $\theta$ then $(C_1\theta - L_1\theta) \vee (C_2\theta - \neg L_2\theta)$ (where minus has the syntactic meaning of "not containing") is called a *binary resolvent.*

Thus, for the first-order predicate calculus, the resolution inference rule can be formulated as follows: [2]. If proper most general unifiers exist, from

[2]Different authors present resolution deductive systems in a slightly different form. The above definition originating in [CL73] presents a single inference rules combining both resolution and factoring. It is equally possible to give a deductive system where resolution and factoring are treated as different rules. The way resolution is defined may have implications on completeness of some refinements of resolution. Overview of different formulations appears in [Le97].

$C_1$ and $C_2$ infer: A binary resolvent of $C_1$ and $C_2$, or A binary resolvent of $C_1$ and a factor of $C_2$, or A binary resolvent of a factor of $C_1$ and $C_2$, or A binary resolvent of a factor of $C_1$ and a factor of $C_2$

**Lemma 1 (Lifting Lemma).** *If $C'_1$ and $C'_2$ are instances of $C_1$ and $C_2$ respectively, and if $C'$ is a resolvent of $C'_1$ and $C'_2$, then there is a resolvent $C$ of $C_1$ and $C_2$ such that $C'$ is an instance of $C$ (*see Figure 3.2*).*

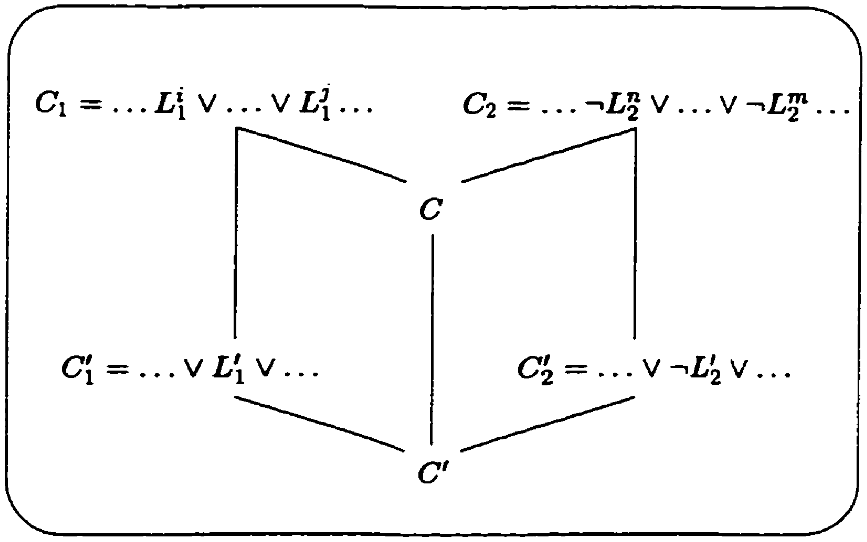
$C_1 = \ldots L_1^i \vee \ldots \vee L_1^j \ldots$
$C_2 = \ldots \neg L_2^n \vee \ldots \vee \neg L_2^m \ldots$
$C$
$C'_1 = \ldots \vee L'_1 \vee \ldots$
$C'_2 = \ldots \vee \neg L'_2 \vee \ldots$
$C'$

Figure 3.2: Lifting

*Proof.* Let $C' = (C'_1\theta - L'_1\theta) \vee (C'_2\theta - \neg L'_2\theta)$ where $\theta$ is the most general unifier of $L'_1$ and $\neg L'_2$.

Take the literals $L_1^i, \ldots, L_1^j$ and $\neg L_2^n, \ldots, \neg L_2^m$ in $C_1$ and $C_2$ which factored into $L'_1$ and $\neg L'_2$ during the transition from $C_1$, $C_2$ into $C'_1$, $C'_2$ and find their most general unifiers $\gamma_1$ and $\gamma_2$. After unification, literals in each group become: $L'_1 = L_1^i\gamma_1 = \ldots = L_1^j\gamma_1$ and $L'_2 = L_2^n\gamma_2 = \ldots = L_2^m\gamma_2$ which are thus literals in the factors $C'_1$ and $C'_2$. Let $\gamma = \gamma_1 \cup \gamma_2$ Since clauses $C_1$ and $C_2$ can be, without loss of generality, chosen with disjoined variable sets: $L_1' = L_1^i\gamma = \ldots = L_1^j\gamma$ and $L_2' = L_2^n\gamma = \ldots = L_2^m\gamma$. Thus $C_1\gamma = C_1'$ and $C_2\gamma = C_2'$.

Since $L_1'$ and $L_2'$ unify with $\theta$, and these are instances of $L_1^i$ and $L_2^n$ respectively, $L_1^i$ and $L_2^n$ should also have a most general unifier $\sigma$ which is more general or equal to a unifier $\gamma \circ \theta$. Thus there exists some $\lambda$ so that $\sigma \circ \lambda = \gamma \circ \theta$.

Thus the resolvent of $C_1$ and $C_2$ is

$$C = (C_1\sigma - L_1^i\sigma) \vee (C_2\sigma - L_2^n\sigma)$$

but $C'$ is an instance of $C$ since (somewhat informally)

$$C\lambda = ((C_1 - L_1^i) \vee (C_2 - L_2^n))\sigma \circ \lambda =$$
$$((C_1 - L_1^i) \vee (C_2 - L_2^n))\gamma \circ \theta = ((C_1\gamma - L_1^i\gamma) \vee (C_2\gamma - L_2^n\gamma))\theta =$$
$$((C_1' - L_1') \vee (C_2' - L_2'))\theta = C'$$

and so $C'$ is an instance of $C$. □

**Theorem 7 (Completeness of Resolution Principle)** *A set $\mathcal{S}$ of first-order clauses is unsatisfiable if and only if there is a refutation of $\mathcal{S}$.*

*Proof.* Let's suppose $\mathcal{S}$ is unsatisfiable and $\mathcal{A} = \{A_1, A_2, \ldots\}$ is the atom set of $\mathcal{S}$. Let $T$ be a complete semantic tree of $\mathcal{S}$. By Herbrand's theorem $T$ is closed.

Use the following procedure to find a false clause for every node of $T$ other than the failure nodes which must already have associated false clauses since $T$ is closed. Let's suppose that $N_1$ and $N_2$ are neighbors with a common parent $N$ and there are two ground instances $C_1'$ and $C_2'$ of $C_1$ and $C_2$ false in $I(N_1)$ and $I(N_2)$ respectively, but both $C_1'$ and $C_2'$ are not false in $N$. Since $N_1$ and $N_2$ are neighbors in the complete semantic tree, $C_1'$ must contain a literal $L_{n+1}$ and $C_2'$ a literal $\neg L_{n+1}$ hence, by resolution, we can produce a clause $C' = (C_1' - L_1') \vee (C_2' - \neg L_2')$ which is false in $N$ since both $C_1'$ and

$C_2'$ are false in $N$. By Lifting lemma if the ground instances $C_1'$ and $C_2'$ are resolvable so should be the clauses $C_1$ and $C_2$ from the set $\mathcal{S}$.

Using this strategy of folding up the semantic tree, an empty clause can eventually be found for the root node, since at every step we reduce the number of nodes in the tree.

Conversely, suppose a refutation exists. Assume $\mathcal{S}$ is satisfiable, therefore having a model. Since the empty clause is logically implied by $\mathcal{S}$, $M$ should satisfy the empty clause which is a contradiction. □

The procedure used in the proof above can be illustrated by the following example:

**Example 5** *Given a set of clauses:* $C = \{A(x) \vee \neg B(a()), \neg A(x), B(x)\}$ *build a semantic tree and construct corresponding resolution-refutation. The Herbrand universe of* $C$ *is* $\mathcal{U} = \{a()\}$. *The ground atom set is thus* $\mathcal{A} = \{A(a()), B(a())\}$.

*A closed semantic tree can be constructed for this clause set (*see Figure 3.3*).*

*From the semantic tree, using the strategy outlined in the proof of completeness theorem, obtaining the resolution-refutation proof.*

*Since clauses* $A(a()) \vee \neg B(a())$ *and* $B(a())$ *fail in the neighboring nodes of the tree, there must be a resolution of their corresponding clauses from the set* $C$. *Indeed*

$$A(x) \vee \neg B(a()) \text{ and } B(x) \text{ resolve to } A(x)$$

*Thus clause* $A(x)$ *must be false in the node above the two failure nodes considered.*

*Since clauses* $\neg A(a())$ *and* $A(x)$ *now fail in the neighboring nodes of the tree, they must resolve. These clauses indeed resolve to a contradiction which is assigned to the root of the tree.*

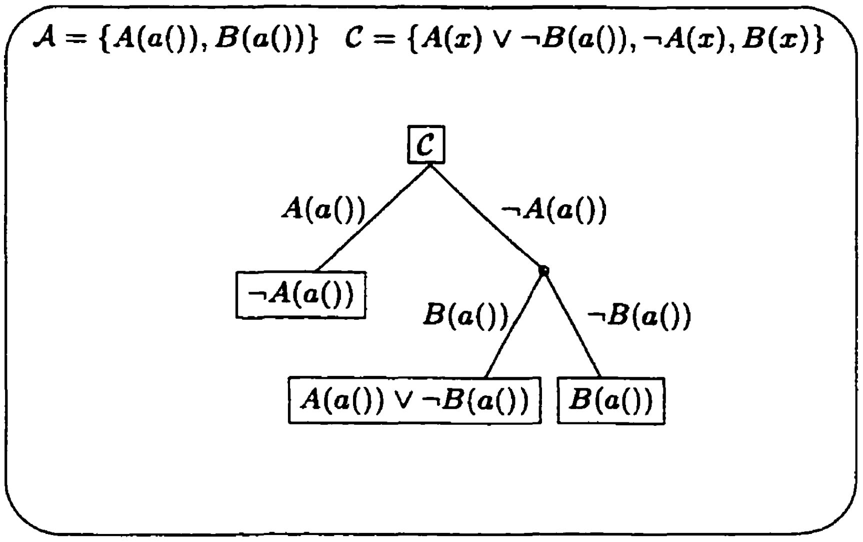
$\mathcal{A} = \{A(a()), B(a())\}$ $\mathcal{C} = \{A(x) \vee \neg B(a()), \neg A(x), B(x)\}$
$\mathcal{C}$
$A(a())$
$\neg A(a())$
$\neg A(a())$
$B(a())$
$\neg B(a())$
$A(a()) \vee \neg B(a())$
$B(a())$

Figure 3.3: A semantic tree for a clause set $\mathcal{C}$.

The completeness theorem shows that if a set of clauses is unsatisfiable, there must exist a resolution-refutation proof and thus it suggests a proof strategy: generating all possible resolvents in an attempt to find a refutation.

# Chapter 4

# Methods for performance improvement

Although, Herbrand's theorem, Davis-Putnam method and Robinson's resolution provide mechanical procedures to automatically prove theorems, such procedures, it is strongly believed, inherently require an exponential (or worse) number of execution steps. Although, it appears, that there cannot be a faster prove procedure in the algorithm complexity sense, for the actual computer implementations to be practical it is necessary to improve performance wherever possible.

An unsophisticated procedure may apply inference rules to the clauses in all possible ways thus often generating redundant or even useless clauses. In many situations, there are refinements available which reduce the number of resolution steps. A *refinement* is called *complete* if it gives a procedure which finds a proof whenever the original procedure was finding one. Some efficient or necessary refinements used on practice may be, however, incomplete and in some cases do not find a proof even when one exists. For instance, all implementations limit the number of retained clauses, size of the retained clauses and allocated proof time. Although such restrictions lead to incompleteness, there is hardly any alternative to using them on practice.

The following sections briefly overview some popular strategy to improve resolution based procedures.

## 4.1 Deletion strategies

In the course of finding a proof, a clause is often produced which is irrelevant for the proof procedure. In other words, the procedure is capable of finding the proof even if such clause was never considered. In many situations, irrelevant clauses can be safely deleted since the completeness properties would not suffer as a result, yet the efficiency of the prover can be considerably improved since the deleted clause will not be used to produce other inferences [Lo78].

### 4.1.1 Pure literal elimination

Recalling that a literal $L$ is said to be *pure* if $\neg L$ does not occur anywhere in the clause set, we can eliminate all clauses containing $L$ from the set which we want to show unsatisfiable. This exactly constitutes the pure literal rule of the Davis-Putnam procedure which was shown to preserve unsatisfiability. In the case of first-order logic if there does not exist a substitution $\theta$ such that $L\theta$ is the same as some other literal $\neg L$ (not counting the sign) then $L$ is pure and can be discarded.

Intuitively, a pure literal does not have a counterpart with which it can be resolved and thus clauses with such literals are irrelevant for the procedure. Although the removal of such clauses can be done at any time, it is most natural to perform this operation as a pre-processing step.

### 4.1.2 Elimination of tautologies

In the course of trying to derive a contradiction from a set of clauses, a clause may be produced which is a tautology (a valid formula being a logical consequence of an empty set of clauses). Such a clause can be safely discarded in the basic resolution-refutation procedure. Let's suppose that we want to show $T \wedge F_1 \wedge \ldots \wedge F_n$ to be unsatisfiable where $T$ is a tautology. Since a tautology is true under any interpretation, for $T \wedge F_1 \wedge \ldots \wedge F_n$ to be unsatisfiable, $F_1 \wedge F_1 \wedge \ldots \wedge F_n$ must be unsatisfiable and thus elimination of $T$ mustn't affect completeness.

Some procedures, such as lock resolution (*see Section 4.2.5*) performing clause ordering, however, require preservation of tautologies to remain complete as they put restrictions on essentially which subsets need to be demonstrated unsatisfiable for showing unsatisfiability of the entire set.

One class of tautologies, which is easy to detect, are clauses of the type $L \vee \neg L \vee F$. Since $L$ must either be true or false in any interpretation $\mathcal{I}$, $L \vee \neg L \vee F$ will be true whenever $L$ is true, but it will also be true whenever $L$ is false (since $\neg L$ will be true in that case).

It may also be fruitful to try detecting other types of tautologies, for instance in theorems with equality a clause $Equal(t,t) \vee F$ will be true under any interpretation due to reflexivity of equality. Depending on the proof procedure and whether equality axioms are represented explicitly or not it may be possible to discard these equational tautologies as well (*see Section 4.3.2*).

### 4.1.3 Subsumption

A clause $B$ ***subsumes*** clause $C$ if and only if $B \Rightarrow C$ is a valid formula (Hence $C$ is a logical consequence of $B$ and is true whenever $B$ is true). The subsumed clause $C$ can be eliminated in some instances when we have the clause $B$, since $B$ is, in a sense, a more general clause. In other instances, subsumed clauses cannot be eliminated, for example, if factoring is used as a distinct rule and not part of binary resolution, the binary factors produced are subsumed by their parents but their elimination will destroy the completeness of resolution-refutation procedure.

A clause $B$ *$\theta$-subsumes* clause $C$, if and only if there exists a substitution $\theta$ such that $B\theta$ is contained in $C$ (in the syntactic sense) and thus the number of literals of $B\theta$ does not exceed the number of literals in $C$. A derived clause $\theta$-subsumed by another retained clause may be eliminated by the resolution refutation procedure without loss of completeness. Thus $\theta$-subsumtion is often referred to simply as subsumption in the framework of the resolution-refutation. Intuitively, the subsumed clause $C$ has more literals then the subsuming clause $B$, thus whenever $B$ is true $C$ must also be true regardless of the interpretation for the remaining literals of $C$ which are not common to $B$ and $C$. Moreover, since the substitution $B\theta$ was contained in $C$, $C$ is, less general then $B$.

The underlining apparatus necessary to compute subsumptions is basically that of one-directional unification also referred to as *matching.* Thus, the implementation of subsumption procedure can reuse some fundamental algorithms, such as unification, already implemented to handle resolution.

For many procedures we can differentiate *forward subsumption* where newly produced clause is eliminated if it is subsumed by a previously deduced clause and *backward subsumption* (more properly referred to as backward subsumtion with replacement) when a test is made if a previously deduced clause is subsumed by a newly produced clause and, in the case when it is true, the old clause is substituted by the new one.

Computations of subsumptions are undoubtedly helpful in reducing the search space, however such computations are quite costly and may slow down the prover to unacceptable degree. Several alternatives are available to avoid this problem. We can either attempt to speed up unification, opt for a weaker variant of subsumption or both. Efficient indexing techniques such as *discrimination-trees* are available for example for forward unit subsumption, a weaker version of general $\theta$-subsumption where the subsumer is a unit clause (*see Section 4.4*) [Ta98]. Similarly, it is relatively inexpensive to compute $\theta$-subsumptions where $\theta$ is an empty substitution. The latter is referred to as *simple subsumption* [Ne97].

## 4.2 Restriction strategies

Although the deletion strategy can often discard some irrelevant clauses, execution time has already been spent on producing them and running the deletion procedure. The restriction strategies work to prevent generation of some redundant clauses thus narrowing the search space. Some strategies also restructure the search in a more convenient way for practical implementations.

### 4.2.1 Linear refinments

*Linear refinements* constrain resolution so that a newly obtained clause is always used to produce the next clause of the deduction either by factoring or binary resolution against some earlier clause or a base clause. This approach was first introduced by Loveland in 1970 and independently by Luckham also in 1970. The simple shape of the deductions produced by linear procedures is a definite advantage for practical implementations which can essentially rely upon *depth-first search*. Thus, with a linear procedure, deductions up to a certain depth are explored one at a time as opposed to computing them all for each given depth before proceeding deeper using *breadth-first search* procedure. The latter method is often referred to as *level saturation technique*. It appears, that no successful implementation can rely exclusively on breadth-first search due to its excessive memory demands nor on depth-first search as the latter may pursue unfruitful deduction for long time. Combinations of the two are more promising such as *iteratively-deepening depth-first search* which explores the tree in a depth-first fashion up to given depth limit $N$ before restarting the search for a deeper limit of $N + 1$. This search combines low memory demand of depth-first search and fairness of breadth-first search and although iteration $N + 1$ repeats entirely all work of iteration $N$, due to exponential nature of the search space, it is only a fraction of work of the deeper iteration.

Basic linear resolution is complete. This fact can be shown by demonstrating a procedure to transform a proof of an arbitrary form into linear form. Such a procedure works by selecting some linear path in the tree representing a proof and pushing the clauses off the path onto the path (*see Figure 4.1 and Figure 4.2*).

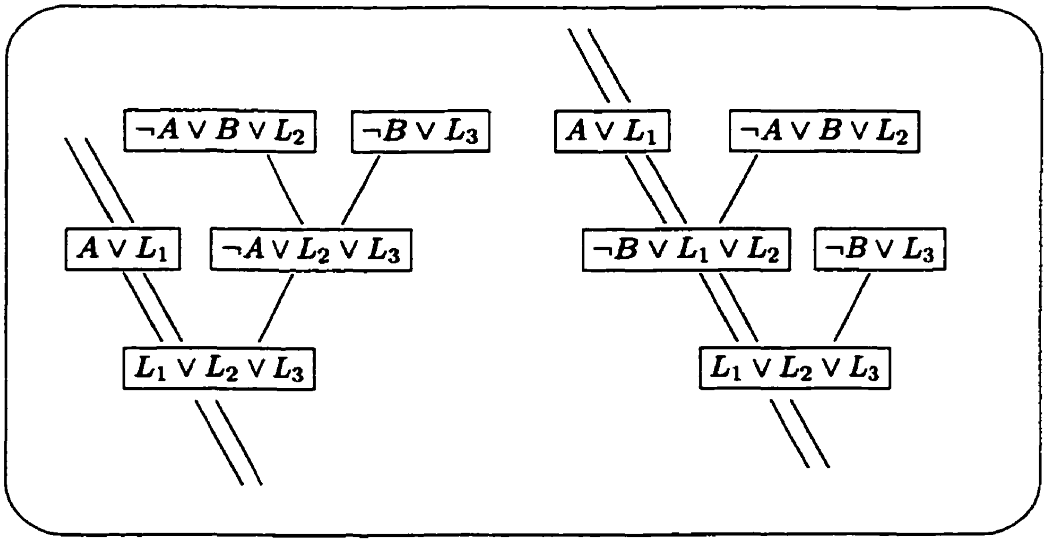

Figure 4.1: Transforming a proof into linear form (case A)

As Figures 4.1 and 4.2 illustrate, there are two cases to consider: a clause violating linear constraint is a binary resolvent of some clauses (*case A*) or it is a factor of some binary resolvent (*case B*) [Ne97]. Thus, if there exists a proof of some arbitrary form this constructive procedure finds some linear proof which thus exists.

The linear refinement can be constrained even further. *Linear-merge refinement*, in order to produce a new clause, uses previously deduced clause together with either a "base" clause from the original set $C$ or a *merge* clause

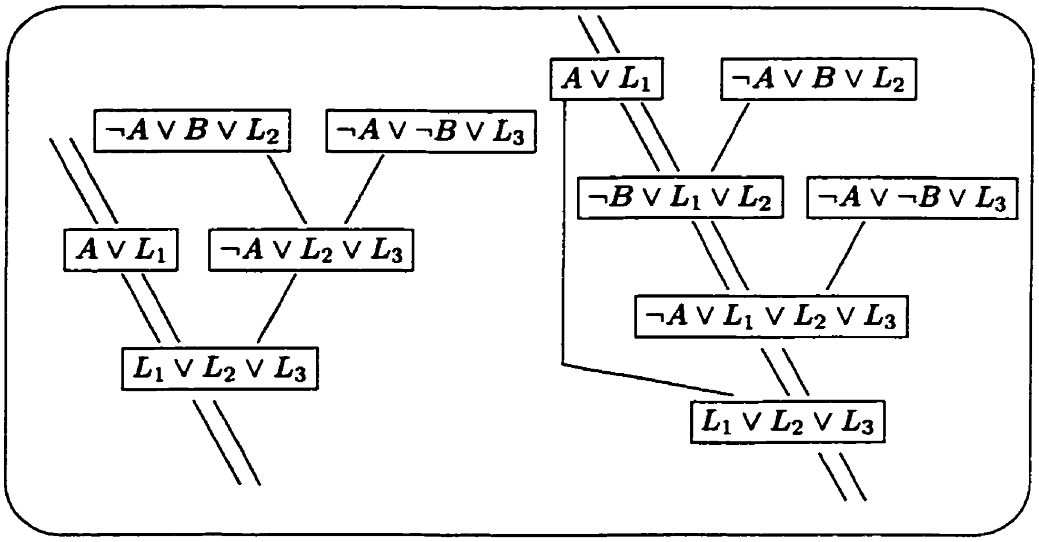
$\neg A \vee B \vee L_2$
$\neg A \vee \neg B \vee L_3$
$A \vee L_1$
$\neg A \vee L_2 \vee L_3$
$L_1 \vee L_2 \vee L_3$
$A \vee L_1$
$\neg A \vee B \vee L_2$
$\neg B \vee L_1 \vee L_2$
$\neg A \vee \neg B \vee L_3$
$\neg A \vee L_1 \vee L_2 \vee L_3$
$L_1 \vee L_2 \vee L_3$

Figure 4.2: Transforming a proof into linear form (case B)

(that is a factor of a binary resolvent) obtained anywhere along the deduction line. This is a more restrictive strategy compared with simple linear strategy which can use either a base clause or any other clause obtained on the deduction line previously. Linear-merge strategy can also be shown to be complete by demonstrating a procedure which given a linear proof constructs a linear-merge proof.

A *linear-unit refinement* constrains the linear search to necessarily use a unit clause (one literal clause) as one of the arguments for binary resolution. This restriction, although quite efficient, leads to incompleteness. For instance from the following unsatisfiable set of clauses we cannot deduce a contradiction using this strategy:

$$C = \{P \vee Q, P \vee \neg Q, \neg P \vee Q, \neg P \vee \neg Q\}$$

Since there is no unit clauses, no resolutions can be performed at all.

A *linear-base refinement*, also referred to as *input refinement*, constrains the binary resolution to necessarily use a base clause from the original set

$\mathcal{C}$ as one argument for the binary resolution. This refinement also leads to incompleteness. Unit refinement and base clause refinement can be shown equivalent in a sense that any theorem provable by one method can be proven by the other method.

Multiple other restrictions, to be considered further, can be used to augment linear resolution which remains one of the more important techniques for practical implementations.

### 4.2.2 Set-of-support

The *set-of-support refinement* or *negated conclusion refinement* was first proposed by Wos, Robinson and Carson in 1965. The important observation underlining this strategy is that with most resolution-refutation methods to prove a theorem $T$ we are attempting to show that $\neg T$ is unsatisfiable. Since the conjunction of all axioms is satisfiable (or at least assumed to be so), we may want to avoid producing deductions coming entirely from the clauses representing the axioms since a contradiction cannot be produced from such a set.

Thus a subset $\mathcal{N}$ of set of clauses $\mathcal{C}$ is called a *set-of-support* if $\mathcal{C} - \mathcal{N}$ is satisfiable. A set-of-support resolution is a resolution of two clauses so that not both are in $\mathcal{S} - \mathcal{N}$. It is not difficult to show that this restriction is complete.

This strategy can be used to augment the linear resolution refinement. Although, it may appear that set-of-support strategy cannot worsen the performance, it is not always so. In some situations lemmas produced by resolving clauses representing axioms lead to shorter proofs, and these will not be obtained with set-of-support refinement.

### 4.2.3 Semantic resolution

Semantic resolution was first introduced by Slagle in 1967. It generalizes Robinson's hyperresolution (*see Section 4.2.4*) (1965) and the set-of-support strategy (*see Section 4.2.2*) proposed by Wos, Robinson and Carson (1965).

Semantic resolution uses an arbitrary interpretation $\mathcal{I}$ to subdivide the clause set into two subsets based on clauses' truth value in that interpretation and prohibits resolutions of clauses coming entirely from the same subset. It additionally uses an arbitrary lexicographical ordering of predicate symbols to limit the possible resolutions.

Given an interpretation $\mathcal{I}$ and the ordering of predicates $\mathcal{P}$, a finite set of clauses is called a *semantic $P - I$ clash*, if and only if the $\mathcal{E}$ clauses (called the electrons) and the $\mathcal{N}$ clause (called the nucleus) satisfy the following conditions:

- The electrons are false in $\mathcal{I}$.
- For each $i = 1, \ldots, n$ there is a binary resolvent or a binary factor of $R_i$ or $R_{i-1}$ and $\mathcal{E}_i$ and assuming that $R_1 = \mathcal{N}$.
- The literal which is resolved upon in every predicate is the largest according to ordering $\mathcal{P}$ among all literals of that electron.
- $R_n$ (called the $P - I$ resolvent) is false in $\mathcal{I}$.

**Example 6** *If*

$$\mathcal{S} = \{\neg A \vee C \vee B, \neg A \vee B, \neg C \vee \neg A, \neg B\}$$

*and*

$$\mathcal{I} = \{A, B, C\}$$

*with ordering $\mathcal{P}$ such that*

$$A \succ B \succ C$$

*then $\neg A$ is a $P - I$ resolvent of $P - I$ clash $\{\neg C \vee \neg A, \neg B, \neg A \vee C \vee B\}$*

It is important that the order of electrons does not matter in the clash so that the same $P - I$ resolvent will be produced for any ordering of the electrons. As Figure 4.3 illustrates $\neg A$ was produced in both cases of resolving the shown clash.

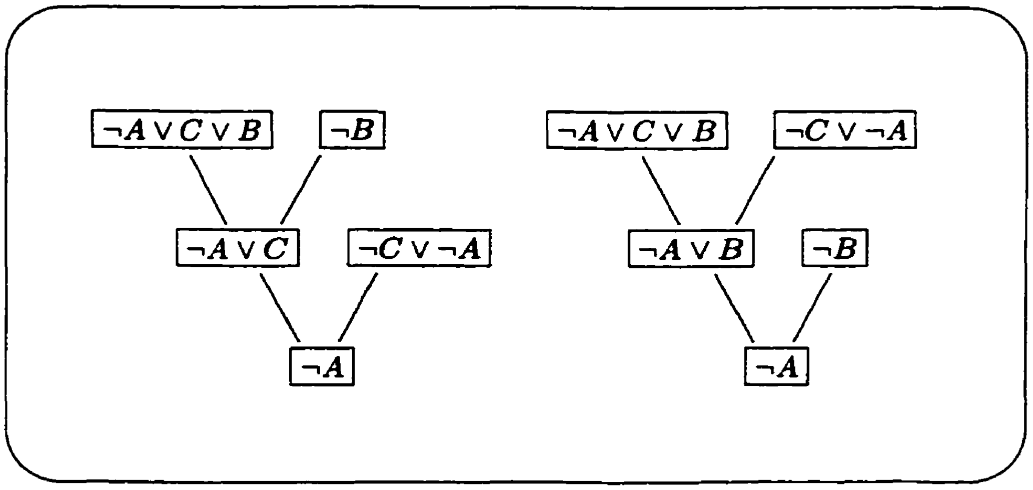

Figure 4.3: Irrelevance of electron's order in a clash

This property is important as it allowed to never explicitly generate the intermediate clauses $\neg A \vee \neg B$ and $\neg A \vee C$ which, as this example illustrates, may well be different and depend on the chosen order of binary resolutions.

A proof is called a *$P - I$ proof* if its every clause is either a base clause or a $P - I$ resolvent.

The restrictions of the semantic resolution can be shown to be complete. Any interpretation can be used for clause subdivision and since unsatisfiable sets of clauses do not have models, the subdivision using any interpretation will produce two non-empty subsets of clauses. Intuitively, the semantic resolutions guide the procedure towards locating a contradiction since the $P-I$ resolvents must be false in the selected interpretation and the contradiction is false in any interpretation.

The semantic resolution strategy generalizes over the strategies of hyperresolution and the strategy of set-of-support which are obtained when particular interpretations are used.

### 4.2.4 Hyperresolution

Let's suppose that in a semantic resolution procedure (*see Section 4.2.3*) the interpretation $\mathcal{I}$ is such that every literal in it is negated. If such an interpretation is used, all electrons and all $P - I$ resolvents must contain only non-negative literals.

A *positive hyperresolution* is a special case of $P - I$ resolution when all literals in the interpretation $\mathcal{I}$ are negative (hence all electrons have only positive literals).

Similarly, if $\mathcal{I}$ contains only non-negative literals, the electrons and $P - I$ resolvents will contain only negative literals.

Thus, *negative hyperresolution* is a special case of $P - I$ resolution when all literals in the interpretation $\mathcal{I}$ are positive (hence all electrons have only negative literals).

### 4.2.5 Clause ordering and locking

Ordering of literals helps to limit the number of possible resolutions. Semantic resolution generally rely upon ordering of literals when constructing the clashes. Although straightforward in the propositional case, the situation is more complex in the case of theorems in first order predicate calculus. Since same predicates may well be applied to different terms, the literal ordering in the propositional sense become ambiguous. As a consequence for semantic resolution, clashes may resolve not to a unique clause but to multiple clauses.

One approach to ordering is to consider clauses as a sequences of disjuncted literals rather than sets of disjuncted literals. Thus, each literal's order corresponds to its place in the clause, resolution is allowed for the biggest literal only and the binary resolvents are obtained by concatenation of sequences of literals representing the ordered clauses and further deleting the literals resolved upon. This method, although potentially efficient, leads to incompleteness when used in conjunction with the semantic resolution procedure. It can also be used in a linear procedure so that we always resolve the least literal in the clause most recently deduced.

Lock resolution is a further refinement of linear resolution-refutation procedure where we assign to each literal an integer representing literal's order. Different occurrences of the same literal will have different order. Resolution is permitted on literals of lowest order. The main difference with the previous ordering is that the literals in the resolvent inherit their order indices from the parent clauses. A merged literal in a factor takes upon the lesser order index of the two literals which were merged together. Lock resolution appears to be an efficient refinment which is complete when used in a basic resolution-refutation procedure. It is incompatible however with the tautology deletion strategy. Using these together destroys the completeness property.

### 4.2.6 Model elimination

Model elimination procedure not only restricts the linear resolution procedure, but also changes some basic aspects of resolution. This method was first proposed by Loveland in 1968 [Lo78] and it is essentially equivalent to the SL resolution procedure proposed by Kowalski and Kuehner in 1971. This procedure also uses the concept of ordered clauses additionally maintaining information on the previously resolved literals which allows us to restrict binary resolutions so that its one argument is always a base-clause. The information on the resolved literals which it maintains roughly corresponds to the information contained in merge clauses obtained before on the deduction line with which it was also necessary to resolve in linear-merge refinement to preserve completeness. In fact, it can be shown that every model elimination deduction can be transformed into linear-merge deduction and thus demonstrate that model-elimination procedure is complete.

The information on the resolved literal is preserved in the ordered clause as a bracketed literal. For instance if $P \vee Q$ is resolved against $\neg Q \vee R$ (the last literal in the first clause must be resolved upon) we normally obtain $P \vee R$ as an ordered resolvent. For this procedure however we want to preserve the literal $Q$ and hence we represent the resolvent as: $P \vee [Q] \vee R$.

Model elimination procedure operates with three inference rules:

- **Extension:** Performs ordered binary resolution with retention of the resolved literal as a bracketed literal.
- **Reduction:** If the last literal in the ordered clause unifies with some bracketed literal it is deleted.
- **Factorization:** Performs ordered factoring (so that the last occurrence of the literal to be merged is deleted from the clause).

Additionally, all bracketed literals which are at the tail of the clause, not followed by an unbracketed literal, are deleted after every step.

For instance, the set of ordered clauses given in the following example can be shown unsatisfiable by model-elimination procedure. As with any strategy maintaining ordered clauses, we start with the first clause attempting to resolve its last literal and continue at each consecutive step to resolve the last literal of the clause obtained at the previous step.

**Example 7** *Base clauses: 1:* $P \vee Q$, *2:* $P \vee \neg Q$, *3:* $\neg P \vee Q$, *4:* $\neg P \vee \neg Q$

*Proof:*

| | |
|---|---|
| *5:* $P \vee [Q] \vee P$ | *extension of 1 with 2.* |
| *6:* $P$ | *factoring of 5, deletion of tailing literal.* |
| *7:* $[P] \vee Q$ | *extension of 6 with 3* |
| *8:* $[P] \vee [Q] \vee \neg P$ | *extension of 7 with 4* |
| *9:* $[]$ | *Reduction of 8, deletion of tailing literals* |

The procedures involving the clause trees [HS96], where we also maintain the information on the resolved literals as internal nodes of a tree representing a clause, are, in many respects similar to the model-elimination procedure.

### 4.2.7 Weighting heuristics

Beside strict restriction strategies, there exist numerous heuristic strategies which seem to improve the efficiency of the prover on practice but are not guaranteed to do so.

One of the more popular strategies of this type is that of *unit preference.* With this strategy we are not required to always take a unit clause as one of the arguments for binary resolution (which was the case with unit refinement) but we would simply prioritize this kind of resolutions. Since the contradiction is a clause of length zero and since unit resolutions always decrease the size of clauses such strategy appears to be very helpful on practice.

Generalizing on unit preference, we can introduce a weighting strategy so that clauses of lesser weight are prioritized by the resolution. A popular weighting criteria is the number of literals in a clause. Similarly to the unit preference, shorter clauses have probably more potential on practice to lead to the contradiction.

Linear procedures employing iteratively-deepening depth-first search can be modified to look beyond the current depth limit if there are resolutions which decrease the weight of the current clause. This *extended search method* may help to avoid extra iterations by pursuing chains of resolutions likely to lead to the contradiction even when it is beyond current depth limit.

## 4.3 Specialization strategies

Some important subclasses of theorems admit special refinements allowing for efficient implementations. Some of these refinements may sacrifice completeness in the general case but can be shown complete for some class of theorems.

### 4.3.1 Sets of Horn clauses

Linear-unit refinement is a constrained version of linear resolution (*see Section 4.2.1*) which demands that one of the arguments for the binary resolution be a unit clause. Although incomplete for derivation of a contradiction from an arbitrary set of clauses, this refinement is complete for a subset thereof, namely any set of Horn clauses. A *Horn clause* is a disjunction of literals such that it contains at most one positive literal.

Although, not all formulas can be represented as Horn clauses, many problems can be formulated using the latter since a Horn clause represents an implication of a conjunction of literals which is a natural representation of logic statements for many situations. Naturally the following implication of a conjunction

$$(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \Rightarrow G$$

if normalised will be transformed first into

$$\neg(F_1 \wedge F_2 \wedge \ldots \wedge F_n) \vee G$$

and finally into a Horn clause

$$\neg F_1 \vee \neg F_2 \vee \ldots \vee \neg F_n \vee G$$

Linear-unit refinement is equivalent in strength to linear-base refinement which demands that one of the arguments of binary resolution be a base-clause. Thus linear-base refinement is also complete in the case of a set of Horn clauses.

The Prolog programming language relies on Horn clauses for representation of theorems and its implementations commonly use base-clause refinements for solving problems [St88].

### 4.3.2 Theorems with equality

Equality predicate is an inherent part of theorems in various areas of mathematics. A single predicate of equality, a single function of succession and a constant zero can formalize the number theory. The properties of equality namely: reflexivity, symmetry, transitivity and substitutability allow specific refinements reducing the number and the complexity of general resolution-refutation [CL73].

Although the theorems with equality can be solved by regular methods, the four properties of equality would have to be represented by axioms. Thus, the following three axioms would have to be added to represent reflexivity, symmetry and transitivity:

$$Equal(x, x)$$

$$Equal(x, y) \Rightarrow Equal(y, x)$$

$$Equal(x, y) \wedge Equal(y, z) \Rightarrow Equal(x, z)$$

To insure substutivity, however, every function and every predicate employed will have to have a clause per every unit of its arity such as the following example of substitutivity axiom for a binary function $f(x, y)$:

$$Equal(x, y) \Rightarrow Equal(f(x, z), f(y, z))$$

$$Equal(x, y) \Rightarrow Equal(f(z, x), f(z, y))$$

This is a consequence of the fact that substitutivity axioms in its original form were of higher order and quantified over all individual predicates and all individual functions. This necessary approach increases considerably the number of axioms and causes generation of many useless clauses by the proving procedure.

To remedy this situation special inference rules treating the equality internally can be introduced. One of such rules is *paramodulation* first described by Robinson and Wos in 1969. Use of this inference rule allows us to avoid introducing equality axioms and it considerably shortens lengths of deductions by taking advantage of the properties of equality, particularly substitutivity.

From the theoretical perspective such inference rules as paramodulation restrict consideration to a subset of all interpretations (In this particular case to the set of $E$-interpretations, that is all interpretations of the equality theory – these which satisfy the reflexivity, transitivity, symmetry and substitutivity axioms).

The paramodulation inference rule can be described as follows.

Let $L[t] \vee F_1$ and $Equal(r, s) \vee F_2$ be two clauses where $t, r, s$ are terms and $L[t]$ is a literal depending on term $t$. If $t$ and $r$ have a most general unifier $\theta$ then from $L[t] \vee F_1$ and $Equal(r, s) \vee F_2$ we can infer: $L[s\theta] \vee F_1\theta \vee F_2\theta$. The inferred clause is called a *binary paramodulant* from $Equal(r, s) \vee F_2$ into $L[t] \vee F_1$.

**Example 8** *The clause $Equal(a(), b()) \vee R(b())$ can be paramodulated into $P(x) \vee Q(x)$ directly producing: $P(b()) \vee Q(b()) \vee R(b())$ without any intermediate clauses which would have to be produced by unrefined binary resolution in this case.*

Using both resolution and paramodulation creates a complete strategy for theorems with equality.

It must be noted that introducing paramodulation inference rule alone does not allow to discard all equality axioms. Particularly, the axiom of reflexivity: $Equal(x, x)$ must still be preserved to retain completeness. This axiom allows us to resolve with some other negative equality literal whose terms of equality may be made the same under some substitution. It is possible to augment paramodulation inference rule with another one, often called *identity assertion* allowing to perform inferences as described above and then discard all equality axioms and preserve completeness.

The paramodulation inference rule can be further refined and restricted along very similar lines with that of binary resolution. For instance, beside discarding tautologies of the type $L \vee \neg L$ we can also now discard equational tautologies of the type $Equal(t, t)$. Furthermore, hyperresolution (in a somewhat weaker form) and linear strategies can be extended with paramodulation. The former is referred to as *hyperparamodulation*. This strategy is using an ordering $\mathcal{P}$ of the predicate symbols and demands that the two clauses to be resolved are positive and the literals paramodulated upon contain the largest predicate symbol according to the ordering $\mathcal{P}$. Also possible are strategies of linear-merge, linear-unit and linear-base paramodulation.

In many cases, the equality theorems lead to complex clauses which represent essentially the same information as previously retained clauses but in a more involved way. Somewhat like subsumption, the demodulation procedure uses unit equality clauses referred to as *demodulators* to rewrite derived clauses into simpler form. Consider the following example.

**Example 9** *$Greater(sum(0(), x), y)$ can be simplified into more meaningful $Greater(x, y)$ in the presence of a demodulator $Equal(sum(0(), x), x)$.*

For a unit equality clause to become a demodulator we must insure that the transformation it will cause is simplifying. The criteria for that is that left term of equality is longer then the right term of equality and that every variable that occurs in the right term must also occur in the left term.

Combining multiple refinements allows for efficient strategies to handle theorems with equality which are very common in multiple areas of mathematics. Multiple theorems can even be stated as pure equality theorems which only have the equality predicate. These equational theorems can be proven by demodulation alone used as a rewrite rule without involving binary resolution directly [Mc94].

It must be noted that especially for linear procedures, introduction of paramodulation does not guarantee improved performance in all cases. Paramodulation tends to shorten the deductions so that it can produce an inference in one step whereas without it such an inference could only be produced through one or several resolutions with equality axioms. Although this may indeed lead to a shorter proof it also has an undesirable effect of increasing the fan-out of the search space, so that it grows considerably already at relatively shallow depth. This may cause proof procedures (especially these based on linear refinements) to actually slow down.

## 4.4 Retention strategies

A linear proof procedure examines the search space in a depth-first fashion. Thus, clauses generated on one path will be discarded during backtracking. Some of those clauses may be very promising and should perhaps be retained. Obvious candidates for retention are unit clauses, since a resolution with a unit clause always shortens the length of a current clause, and demodulators – simplifying unit equalities which allow to reduce the length of matching terms in clauses encountered elsewhere.

Since there may be a large number of such clauses to retain, and, what's more important, a newly derived clause may have to be checked against all retained clauses, it becomes important to consider efficient storage and retrieval data structures. Among such efficient means of clause retention are the hash tables and various indexing techniques such as discrimination trees.

### 4.4.1 Hash tables

A hash table is an array each cell of which may contain a hash code of some element so that the index of the element in the hash table is determined by its hashcode. Hashcodes are computed in such a way so that different elements hopefully receive different codes. With such a data structure checking if a certain literal is retained by the table is essentially an array look up - an $\mathcal{O}(1)$ operation.

On obtaining a unit clause, its hash code is computed and inserted into the hash table. It is assumed that the sign of unit's literal does not affect the hashcode, yet is recorded as perhaps an additional bit. This procedure immediately allows us to find unit contradictions: if such a code already exists in the table and the sign is opposite - a contradiction is found.

Retention of units in a table allows us to potentially reduce number of literals in the clause to be generated elsewhere in the search. When a new clause is infered, we can compute the hash code for its every literal and perform hash table look ups. If a hash table hit results and the sign of the element stored in the table is different, the current literal can be deleted as it would have resolved with the unit clause stored in the table. If the sign is the same, however, the unit clause stored in the table subsumes the current clause and thus, all literals with the exception of the current one can be discarded from the current clause.

It must be noted that by storing only a hashcode for a unit clause we lose all the information on the structure and the history of the unit clause itself. Hence, it becomes more difficult to restore the entire proof (we may have to restart the search to find some unit clauses which were stored in the hash table and contributed in the refutation). We must also verify the proofs, elements of which used hash table look ups. Due to potential of hash errors when different literals got the same hashcodes, some obtained proofs may be invalid.

What's worse, however, is that by losing the structure of the unit clauses, these can only be used to resolve in the cases with an empty substitution when unit's literal exactly matches some literal in the current clause. This, of course, is only a tiny fraction of all possible resolutions whose substitution is not empty. To partly relieve this problem, for every unit clause, we can also retain certain number of its variants where variables were substituted by some constants. This is likely to result in a higher hit ratio, yet it also increases the load of the hash table.

Beside the use of hash tables for unit resolution and unit subsumption, there are multiple other aspects of theorem provers where this data structure can be employed. For instance it can be used to reduce search redundancy. Since it is expensive (although some times necessary in the framework of the linear refinements) to search the same clause multiple times if we arrived to it by different path, we may hash all clauses found and not research a clause which causes a hash table hit, i.e. a clause which was searched before.

### 4.4.2 Discrimination trees

A *discrimination-tree* index is a *trie* like data structure that represents collectively the structure of all terms inserted into the index. A downward path in this tree from the root to the leaf describes the structure of a single term. It can serve as a pre-filter to unification or matching, thus allowing to retain unit clauses for unit resolution, unit subsumers or left terms of unit equalities for demodulation [Mc93].

The discrimination tree look up traverses and backtracks through both the query term and the tree and finds the terms which have a potential to unify or match with the query term.

Every node in the tree, with the exception of its root, is labeled with either a functor, a constant or a special symbol "$*$" to describe any variable. if two terms have some common prefix, their paths in the tree will be the same for the length of the prefix.

In the discrimination tree represented in Figure 4.4 the terms $g(b(), a())$, and $g(b(), c())$, are represented by a common path $g - b()$ describing common prefix at which point the paths for the two terms branch ending with $a()$ in one case and with $c()$ in the other.

Thus, with this method, it is possible to compress structure of many similar terms into a relatively compact data structure which also enables fast querying to determine all terms which has a potential to unify.

As it was mentioned, any variable is represented in the tree by a special symbol "$*$". Thus, during querying process any corresponding subterm in the query term will be accepted. For instance, the term $f(x, x)$ is represented in the tree by a path $f - * - *$ but this will also be a representation for $f(x, y)$,

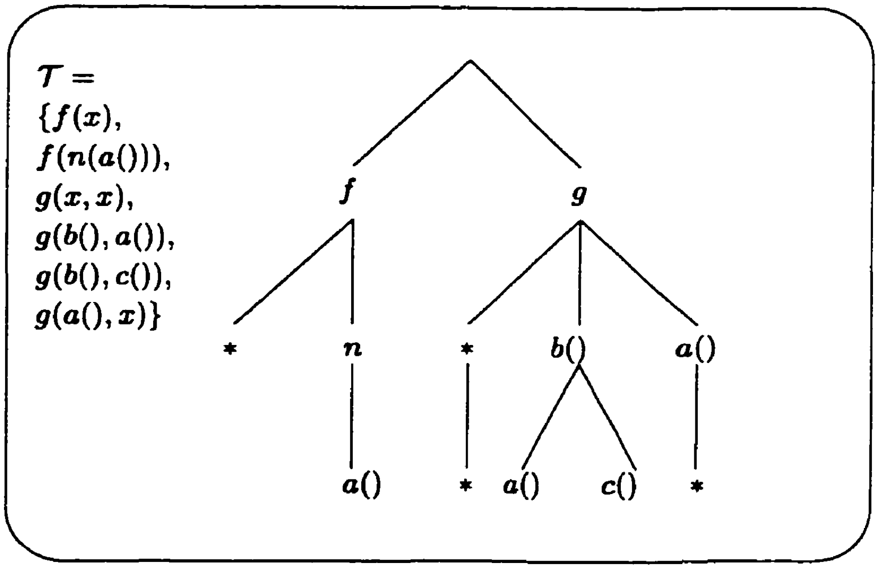

Figure 4.4: A discrimination tree for a term set $\mathcal{T}$.

and obviously some terms (e.g. $f(a(),b()))$ which may unify with the latter will not unify with the former.

A refinement of the basic discrimination tree called *variable-containing discrimination tree* does represent explicit information on variables, and thus allows precise queries. This, however, incurs additional complexity of the data structure and the querying algorithm.

The use of the discrimination tree in a theorem prover is similar to that of a hash table. The advantage however is that since the entire structure of the term is represented, we do not have to store any additional variants of the term where constants were substituted for certain variables.

# Chapter 5

# Approaches for theorems in geometry

Refinements outlined in Chapter 4, if properly applied, should allow us to considerably improve the performance of resolution based theorem provers. The majority of those refinements were generic and applicable to any theorem. Some, however, exploited certain knowledge of the underlining axiom system to find some computational shortcuts. For instance, knowledge of behavior of the equality predicate which possesses the properties of reflexivity, symmetry, transitivity and substitutivity allows us to formulate the inference rule of paramodulation with the internal knowledge of the above properties and delete now unnecessary axioms to describe them. This strategy allows the prover to make relatively complex inferences in one step and not to produce many of often useless intermediate inferences thus potentially speeding up the search.

Other classes of theorems may benefit from such specialization strategies arising out of special properties of their common axioms. One of the oldest domains in mathematics (if not the oldest), where this may be of interest, is Euclidian geometry. It also so happens that theorems in Euclidian geometry are relatively difficult to prove automatically and many generic refinements, which help in other domains, do not seem to be strong enough in this case (As will be observed in section 5.3.2 when describing experiments with two popular provers, many relatively simple (in mathematical terms) theorems were not proven).

By considering the set of axioms it is possible to devise refinements specifically aimed at geometry theorems. Thus, in this chapter we first overview formalizations for Euclidian geometry. Then, multiple refinements will be proposed. Since automated theorem proving is inherently a practical subject where the value of a method should be decided by its performance in practice, these refinements were implemented in GLIDE theorem prover. The prover and the experiments carried out will be described to show how the proposed methods fared against generic strategies.

## 5.1 Formal systems for geometry

Planar geometry holds the distinction of being one of the earliest domains to be formalized. The system of multiple axioms was put forward by Euclid in his "Elements". In many respects this was the beginning of axiomatic method itself. The axioms were:

- A straight line may be drawn from any point to any other point
- A finite straight line can be extended continuously in a straight line
- A circle may be described with any center and any radius
- All right angles are equal to one another
- If a straight line meets two other straight lines so as to make the sum of the two interior angles on one side of the traversal less than two right angles, the other straight lines, extended indefinitely, will meet on that side of the traversal [BB91]

These five basic axioms were used to derive new theorems in a sequential manner, so that for the first theorem only axioms were used whereas proofs of following theorems could use previously proven theorems as well.

In the beginning of this century, with the advances in formal logic, the need reemerged to find formal foundations for different domains of mathematics, the axiom systems had to be expressed as formulas in languages of logic.

An axiomatization for geometry was proposed by Hilbert. It operated on the universe containing points, lines and planes and had predicates to express relations of these individuals.

In 1926 Alfred Tarski proposed a much more compact formalization for Euclidian geometry. It described the universe containing only points and had only two essential predicates besides that of equality: equidistance and betweenness.

Over the years, the latter system was successfully applied to the needs of automated theorem proving. A slightly modified Tarskian axiom system was described by McCharen et al. in [MO76] and later Quaife made some further modifications in his work [Qu89].

Hilbert's system was also used in automated theorem proving research, for instance in [Be92].

Besides, resolution based strategies, other approaches to prove geometry theorems were also developed such as the algebraic method of Wu described in [Ch88]. This latter method is very powerful and with its help it is often possible to prove theorems too difficult for resolution based provers. However, this approach is not very general and can cope only with theorems which can be expressed as equations (i.e. it cannot handle inequalities).

Special refinements were also considered over the years for subdomains of geometry, for instance McCune and Padmanaghan describe a resolution rule for reasoning about cubic curves in [MP96].

Tarski axiomatization was chosen for the purposes of this work due to its popularity, simplicity and convenience as well as because of a large amount of available test theorems based on this system.

### 5.1.1 Tarski-Quaife axiom system

Quaife's edition of Tarski axiom system [Qu89], beside the predicate of equality, is also using two other main predicates: *equidistance* and *betweenness* which will be further denoted as $D$ and $B$ respectively.

Equidistance predicate assumes four arguments - two pairs of points, and it expresses the fact that the distance between points of the first pair is the same as distance between points of the other pair. Predicate of betweenness expects three arguments and it represents the fact that the point described by its middle argument is on the line segment between points described by its outer arguments.

The predicate of equidistance is *reflexive* [1]:

$$D(x, y, y, x)$$

This means that the distance from point $x$ to point $y$ is the same as distance from point $y$ back to point $x$. It is *transitive*:

$$\neg D(x, y, z, v) \vee \neg D(z, v, u, w) \vee D(x, y, u, w)$$

Equidistance possesses property of *identity*:

$$\neg D(x, y, z, z) \vee Equal(x, y)$$

Meaning that if for two pairs of points with equal distances, one pair is trivial (distance from the point to itself) the other pair should be trivial as a consequence.

The predicate of betweenness also possesses property of identity:

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

[1]Tarski axiom system is shown here in clausual form.

Meaning that if some point is between another point and that same point again, all three points must in fact be the very same one.

With the aid of equidistance and betweenness predicates, it is possible to describe more complex axioms. The following one is known as the *Outer five-segments axiom* and it asserts relationships of similar triangles on the plane:

$$\neg D(x,y,x',y') \vee \neg D(y,z,y',z') \vee \neg D(x,v,x',v') \vee \neg D(y,v,y',v') \vee$$

$$\vee \neg B(x,y,z) \vee \neg B(x',y',z') \vee Equal(x,y) \vee D(z,v,z',v')$$

The meaning of this axiom is depicted in Figure 5.1.

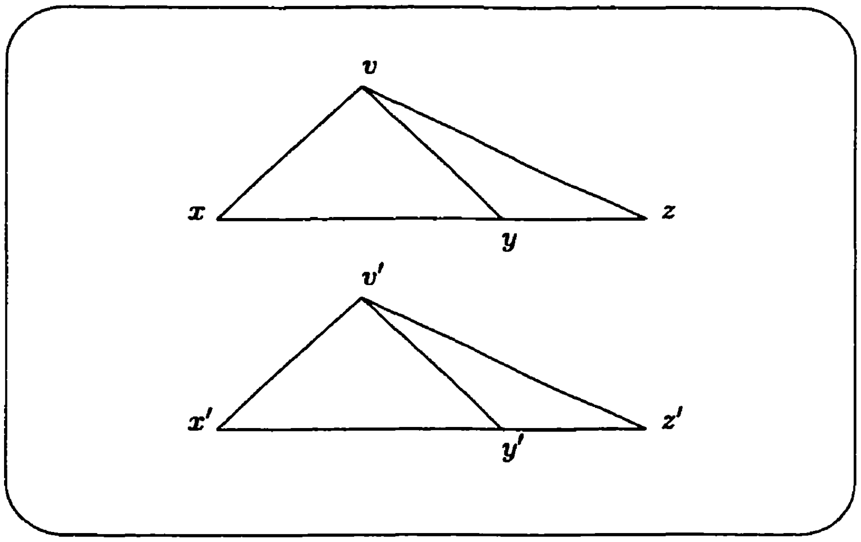

Figure 5.1: Outer five-segment axiom.

Beside predicates of equidistance and betweenness this axiom system also uses several functions. All of these function actually arise during Skolemisation process (*see Section 2.2.3*), when existentially quantified variables are substituted by new constants or function. Since axioms described here are already in normalized, clausual form, functions appear instead of existentially quantified variable to describe existence of some points which depend on some other points.

One of the function used for this purpose is the *Extension* function, to be denoted as $ext(x, y, u, v)$. This function maps its four arguments, interpreted as points, to a point. The first pair of arguments of extension function describes a segment and the second pair specifies the distance by which the segment is to be extended to arrive to the point specified by the extension function. The following axiom (known as the *Segment construction axiom* describes the main property of extension function:

$$B(x, y, ext(x, y, w, v))$$

$$D(y, ext(x, y, w, v), w, v)$$

Another function used is, what's known as, the *Inner Pasch* function, used to describe existence of an intersection point of two segments as shown in Figure 5.2.

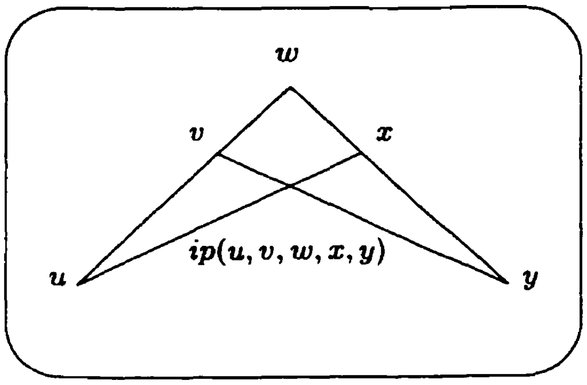

Figure 5.2: Inner Pasch axiom.

The main property of inner Pasch point is represented by the following axiom:

$$\neg B(u, v, w) \vee \neg B(y, x, w) \vee B(v, ip(u, v, w, x, y), y)$$

$$\neg B(u,v,w) \vee \neg B(y,x,w) \vee B(x, ip(u,v,w,x,y), u)$$

Existence of three points not all on the same line is described by the *Lower dimension* axiom using the following three clauses:

$$\neg B(p_1(), p_2(), p_3())$$

$$\neg B(p_2(), p_3(), p_1())$$

$$\neg B(p_3(), p_1(), p_2())$$

Another important geometric property is expressed by the *Upper dimension* axiom which says that if three points are equidistant to two other distinct points, all three must be on the same line. This axiom is expressed by the following single clause:

$$\neg D(x,w,x,v) \vee \neg D(y,w,y,v) \vee \neg D(z,w,z,v) \vee$$

$$\vee B(x,y,z) \vee B(y,z,x) \vee B(z,x,y) \vee Equal(w,v)$$

Existence of a single line parallel to the given line and passing through a point not on the second line is expressed by Euclid's axiom[2]. In Tarskian axiom system this axiom is described with the help of two special functions, denoted as $euc_1(x,y,z,v,w)$ and $euc_2(x,y,z,v,w)$. These two points represent existence of a line passing through them. Three clauses are required to express the axiom:

$$\neg B(u,w,y) \vee \neg B(v,w,x) \vee Equal(u,w) \vee B(u,v,euc_1(u,v,w,x,y))$$

$$\neg B(u,w,y) \vee \neg B(v,w,x) \vee Equal(u,w) \vee B(u,x,euc_2(u,v,w,x,y))$$

$$\neg B(u,w,y) \vee \neg B(v,w,x) \vee Equal(u,w) \vee B(euc_1(u,v,w,x,y), y, euc_2(u,v,w,x,y))$$

The meaning of the above clauses is depicted in Figure 5.3.

[2]Euclid's axiom in this formulation is equivalent to traditional fifth Euclid's axiom which has a special distinction in geometry. Assertion of this axiom accompanied by the other four Euclid's axioms describe Euclidian geometry whereas its negation accompanied with the other four describe a *hyperbolic* or Lobachevski geometry where at least two distinct lines exist which are parallel to a given line and pass through a point not on the second line.

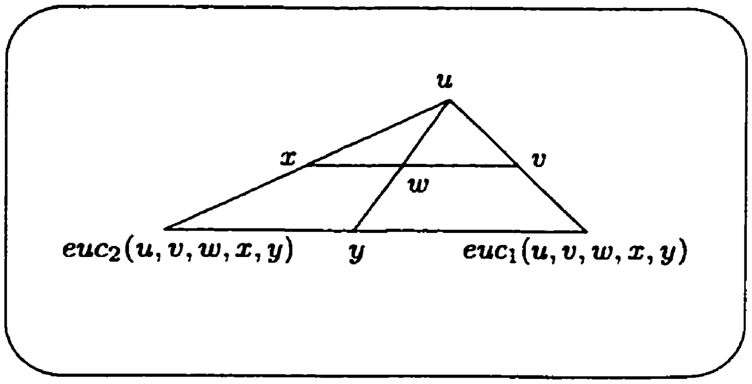

Figure 5.3: Euclid's axiom.

Finally, ***Weakened continuity axiom*** expresses the fact that any segment joining two points one inside, one outside a given circle, intersects the circle. the *Continuity* function denoted as $cnt(x,y,z,u,v,w)$ describes the intersection point. Two clauses are used to describe this axiom:

$$\neg D(u,v,u,v_1) \vee \neg D(u,x,u,x_1) \vee \neg B(u,v,x) \vee \neg B(v,w,x) \vee B(v_1, cnt(u,v,v_1,w,x,x_1), x_1)$$

$$\neg D(u,v,u,v_1) \vee \neg D(u,x,u,x_1) \vee \neg B(u,v,x) \vee \neg B(v,w,x) \vee D(u,w,u,cnt(u,v,v_1,w,x,x_1))$$

Beside the axioms described above, since equality predicate was used, we also require axioms to describe equality: its reflexivity, symmetry, transitivity and its substitutivity with respect to predicates of equidistance and betweenness as well as all functions used: extension, inner Pasch, Euclid's functions and the continuity function.

## 5.1.2 Alternative and additional axioms

Quaife's edition of Tarski axiom system differs somewhat from earlier versions of this axiom system. McCharen et al [MO76] describe a slightly different system which had additional axioms for transitivity and connectivity of betweenness. The axioms to describe these properties were later found to be dependent on other axioms. McCharen's et al axiomatisation represented Pasch axiom in outer rather than in inner form:

$$\neg B(x,w,v) \vee \neg B(y,v,z) \vee B(x, op(w,x,y,z,v), y)$$

$$\neg B(x,w,v) \vee \neg B(y,v,z) \vee B(z, w, op(w,x,y,z,v))$$

It is not hard to see from Figure 5.4 that this axiom is very closely related to the inner Pasch axiom and describe the same geometric property.

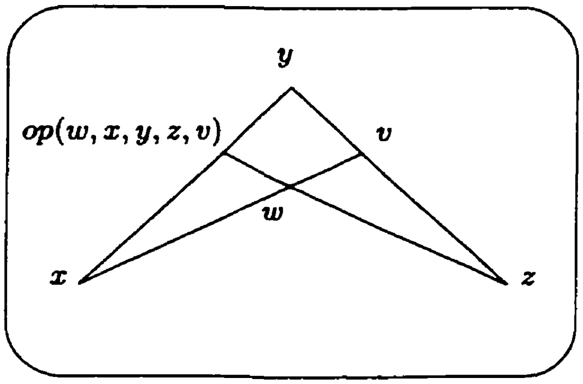

Figure 5.4: Outer Pasch axiom.

There is also a minor change in formulations of Euclid's axiom.

Both Quaife's and McCharen's et al systems weaken the continuity axiom which is, however, admissible for the purposes of elementary geometry. For many purposes these two versions of Tarski axiom system are virtually indistinguishable.

Additional properties of equidistance and betweenness can be derived from the described axioms. For instance symmetry of equidistance and symmetry of betweenness:

$$\neg B(x,y,z) \vee B(z,y,x)$$

$$\neg D(x,y,z,y) \vee D(z,y,x,y)$$

It may be beneficial to add these theorems into the set of base clauses since they may, at times, lead to a shorter proof. It should be noted, however, that there are eight different equidistance congruences and adding seven clauses to describe them all may be overly costly in terms of increased amount of resolvents.

The situation is similar with reflexivity of betweenness:

$$B(x, x, y)$$

$$B(x, y, y)$$

As well as additional reflexivities for equidistance:

$$D(x, y, x, y)$$

$$D(x, x, y, y)$$

These clauses can also be used to augment the base clauses.

It is also convenient to add a predicate of *colinearity* (denoted $C$) to simplify the task of describing theorems. If this is done, it is necessary to specify the properties of this predicate. Most commonly this is achieved by denoting its relationship to the predicate of betweenness, since betweenness describes a case of colinearity.

Thus, if three points are colinear, there are three different possibilities for their betweenness:

$$\neg C(x, y, z) \vee B(x, y, z) \vee B(y, z, x) \vee B(z, x, y)$$

At the same time if a point is between two other points, all three are colinear:

$$\neg B(x, y, z) \vee C(x, y, z)$$

$$\neg B(x, y, z) \vee C(y, z, x)$$

$$\neg B(x, y, z) \vee C(z, x, y)$$

It should be noted that if colinearity predicate is added, it will also be necessary to describe additional equality substitution axioms for this predicate.

## 5.2 Refinements

The difficulty of theorems based on Tarskian-Euclidian axiom system comes from relatively heavy use of equality, quite a large number of clauses, length of the clauses and also from the nature of predicates and functions involved. For instance, predicate of equidistance admits eight different orders of its arguments to describe the very same situation. Indeed $D(a(), b(), c(), d())$ is the same as $D(b(), a(), c(), d())$ or $D(c(), d(), a(), b())$ etc. Imagining that only this potentially results in seven different inferences for every equidistance literal in the current clause, the difficulty of a brute force search becomes obvious.

The refinements of resolution work to delete unnecessary inferences and to prevent some inferences from occurring. By examining the underlining axiom system it may be possible to find refinements to improve performance of the general methods.

### 5.2.1 Generic refinements

Since Tarskian axiom system uses the equality predicate, employing refinements for first-order logic with equality should be beneficial. Particularly, introduction of the inference rule of paramodulation allows us to delete axioms describing symmetry, transitivity and substitutivity of equality. Additional inference rule of identity assertion permits to also delete equality reflexivity axiom (*see Section 4.3.2*). Three clauses are required to describe reflexivity, symmetry and transitivity, one clause for each property. As to substitutivity, one clause is needed for each unit of arity of every predicate and every function. Predicates of betweenness and equidistance have arities of 3 and 4 respectively. Functions of inner Pasch, first and second Euclid, extension and continuity have arities of 5, 5, 5, 4 and 6 respectively. Thus, with this refinement alone, we can reduce the set of base clauses by 35 clauses. Although this may seem like a considerable benefit, unrestricted paramodulation may, at times, generate the number of inferences rivaling what would have been produced by binary resolution with the equality axioms remaining.

The use of unit clause retention for unit resolution and unit subsumption may worsen performance only due to the added cost of storing and querying literals in the data structure. Assuming the use of a data structure with inexpensive complexity for both above operations, gained benefits should greatly exceed the incurred cost.

Adding mechanisms for non-unit subsumption is always more problematic due to the high cost. From practical experience, cheep versions of non-unit subsumption such as s-subsumption contribute to better performance and incur only a reasonable price.

As to many other possible refinements of resolution such as semantic or lock resolution it is unclear to what degree these may help in the case of geometry theorems.

Some of the generic refinements, mentioned above, were employed when testing specialized geometry refinements to be described in following sections. The testing results will be described in Section 5.3.2.

### 5.2.2 Approaches for reflexivities

Both main geometric predicates of Tarskian axiom system: equidistance and betweenness as well as auxillary predicate of colinearity are reflexive. Quaife's version of Tarskian system included the following axiom for reflexivity of equidistance:

$$D(x, y, y, x)$$

Beside this one, there are two easy corollaries which may also have been added to the set of base clauses: *Ordinary reflexivity*: $D(x, y, x, y)$ and *Trivial reflexivity*: $D(x, x, y, y)$.

For a single reflexivity clause $D(x, y, y, x)$ to be removed and completeness maintained, the following inference rule can be introduced:

If a clause $C$ has a negative equidistance literal $\neg D(t_1, t_2, t_3, t_4)$, and this literal's first and last arguments unify with $\alpha$ whereas its second and third arguments unify with $\beta$, i.e. $(t_1\alpha = t_4\alpha)$ and $(t_2\beta = t_3\beta)$, infer $C\alpha \circ \beta - \neg D(t_1, t_2, t_3, t_4)\alpha \circ \beta$ from $C$.

We will refer to this rule as *Equidistance assertion*, and the inference of this rule as *Equidistance assertant.*

It is not hard to show that introduction of this rule permits us to discard the equidistance reflexivity clause while maintaining completeness.

**Theorem 8 (Completeness of equidistance assertion)** *If a clause set $S$ has a resolution proof $P$, then the clause set $S - \{D(x, y, y, x)\}$ will have a resolution-equidistance assertion proof $P'$.*

*Proof.* To obtain $P'$ from $P$ we will use the following procedure: Traverse all inferences of $P$. If $D(x, y, y, x)$ is not among the parents of the current inference, it will also be available on $S - \{D(x, y, y, x)\}$, thus transfer this inference into $P'$.

If $D(x, y, y, x)$ is indeed one of the parents of the current inference, the second parent $C$ must have contained a negative literal $\neg D(t_1, t_2, t_3, t_4)$, and $D(x, y, y, x)$ must have unified with $\neg D(t_1, t_2, t_3, t_4)$ with some substitution $\theta$.

Let's consider the structure of $\theta$. Variable $x$ must have unified with $t_1$ and at the same time it must have unified with $t_4$. Similarly, variable $y$ must have unified with $t_2$ and at the same time with $t_3$. Since $x$ and $y$ are just variables, $\{t_1, t_4\}$ and $\{t_2, t_3\}$ must unify on their own.

Let's suppose that the most general unifier of $\{t_1, t_4\}$ is called $\alpha$ and that of $\{t_2, t_3\}$ is called $\beta$.

The resolvent of $D(x, y, y, x)$ and $C$ is $C\theta - \neg D(t_1, t_2, t_3, t_4)\theta$ but because $C$ does not contain variables $x$ and $y$ this would be the same as $C\alpha \circ \beta - \neg D(t_1, t_2, t_3, t_4)\alpha \circ \beta$ which is exactly an equidistance assertant of $C$ and can thus be transfered into $P'$. $\square$

Equidistance assertion considered here is easily extendible to accommodate ordinary and trivial reflexivities.

The predicate of betweenness is also reflexive. Although this property is not directly represented by the axioms of Tarskian system, it is easily derivable from other axioms. There are two possible reflexivities:

$$B(x, x, y)$$

$$B(x, y, y)$$

An inference rule called *Betweenness assertion*, along the lines of equidistance assertion can be introduced here.

If a clause $C$ has a negative betweenness literal: $\neg B(t_1, t_2, t_3)$, and this literal's first and second arguments unify with $\alpha$, i.e. $(t_1\alpha = t_2\alpha)$, infer $C\alpha - \neg B(t_1, t_2, t_3)\alpha$ from $C$.

The above only accounts for $B(x, x, y)$, but is easily extendible to accomodate $B(x, y, y)$.

It is not hard to see that completeness will not suffer if this inference rule is used and betweenness reflexivity clause (or clauses) are discarded.

The proof of completeness is similar to an analogous proof for equidistance.

In a similar manner, we can also introduce assertion of colinearity. Obviously, since two distinct points define a line, if of three points any two are, in fact, the same, such three points must be colinear:

$$C(x, x, y)$$

$$C(x, y, x)$$

$$C(y, x, x)$$

Introduction of these inference rules has several advantages. It allows us to discard up to eight clauses which were describing reflexivities. This is actually a relatively small advantage since the introduced inference rules mimic resolution with these clauses with a very similar cost. More of an advantage results when some of the reflexivities were absent from the set of base clauses. This may well happen since reflexivity of betweenness is derivable from other clauses. Deriving and then applying reflexivity may lengthen the proof only by a few steps yet every step often comes at a very high and increasing cost.

A weaker version of geometry assertion (i.e. of equidistance, betweenness and colinearity assertion) where all substitutions are empty can be used along with described above general version. For instance, if a clause is encountered which contains a literal $\neg D(t_1, t_2, t_1, t_2)$, it can be immediately removed. This can be done as a post-processing on a clause before it has been used to produce new inferences. It is not difficult to see that completeness will not be sacrificed.

An easy extension to such weaker version of geometry assertion should allow us to easily find geometric tautologies. For instance, if a clause is encountered which contains a positive literal $D(t_1, t_2, t_2, t_1)$, this clause would have been subsumed by equidistance reflexivity axiom and, thus, may be discarded.

### 5.2.3 Approaches for identities

Both main geometric predicates of Tarskian axiom system have identity. The identity for equidistance was expressed by the following axiom:

$$\neg D(x, y, z, z) \lor Equal(x, y)$$

An easy corollary results when the symmetry of equidistance is taken into account:

$$\neg D(x, x, y, z) \lor Equal(y, z)$$

For a single identity axiom $\neg D(x, y, z, z) \lor Equal(x, y)$ to be removed and completeness maintained the following inference rule can be introduced:

If a clause $C$ has a positive equidistance literal: $D(t_1, t_2, t_3, t_4)$, and this literal's third and forth arguments unify with $\alpha$, i.e. $(t_3\alpha = t_4\alpha)$, infer

$C\alpha - D(t_1, t_2, t_3, t_4)\alpha + Equal(t3, t4)\alpha$ from $C$, where "-" has a syntactic meaning of "not containing" and "+" has a syntactic meaning of "including".

We will refer to this rule as *Equidistance identification*, and the inference of this rule as *Equidistance identifier*.

**Example 10** *Given a clause $D(a(), f(b()), x, k()) \vee A(x)$ using equidistance assertion and since $x$ and $k()$ unify, we can infer equidistance identifier $Equal(a(), f(b())) \vee A(k())$.*

It is not hard to show that the introduction of this rule allows us to discard equidistance identity clause while maintaining completeness.

**Theorem 9 (Completeness of equidistance identification)** *If a clause set $S$ has a resolution proof $P$, then the clause set $S - \{\neg D(x, y, z, z) \vee Equal(x, y)\}$ will have a resolution-equidistance identification proof $P'$.*

*Proof.* To obtain $P'$ from $P$ we will use the following procedure: Traverse all inferences of $P$. If $\neg D(x, y, z, z) \vee Equal(x, y)$ is not among the parents of the current inference, it will also be available on $S - \{\neg D(x, y, z, z) \vee Equal(x, y)\}$, thus transfer this inference into $P'$.

If $\neg D(x, y, z, z) \vee Equal(x, y)$ is indeed one of the parents of the current inference, there are two cases to consider.

First case occurs if the second parent $C$ had a positive equidistance literal $D(t_1, t_2, t_3, t_4)$ which resolved with equidistance literal of identity axiom with substitution $\theta$.

If that's the case, variable $z$ must have unified with $t_3$ and at the same time with $t_4$. Since $z$ is just a variable $t_3$ and $t_4$ must have unified on their own.

Let's call the most general unifier of $\{t_3, t_4\}$ as $\alpha$.

The resolvent of $C$ and $\neg D(x, y, z, z) \vee Equal(x, y)$ is obtained as $C\theta - \neg D(t_1, t_2, t_3, t_4)\theta + Equal(x, y)\theta$ but since $x, y$ and $z$ are variables it will be the same as $C\alpha - \neg D(t_1, t_2, t_3, t_4)\alpha + Equal(t_1, t_2)\alpha$ which is an equidistance identifier of $C$ and which can, thus, be transfered into $P'$

Second case occurs if the second parent $C$ had a negative equality literal $\neg Equal(m_1, m_2)$ which resolved with equality literal of identity axiom with substitution $\gamma$ (*see Figure 5.5*).

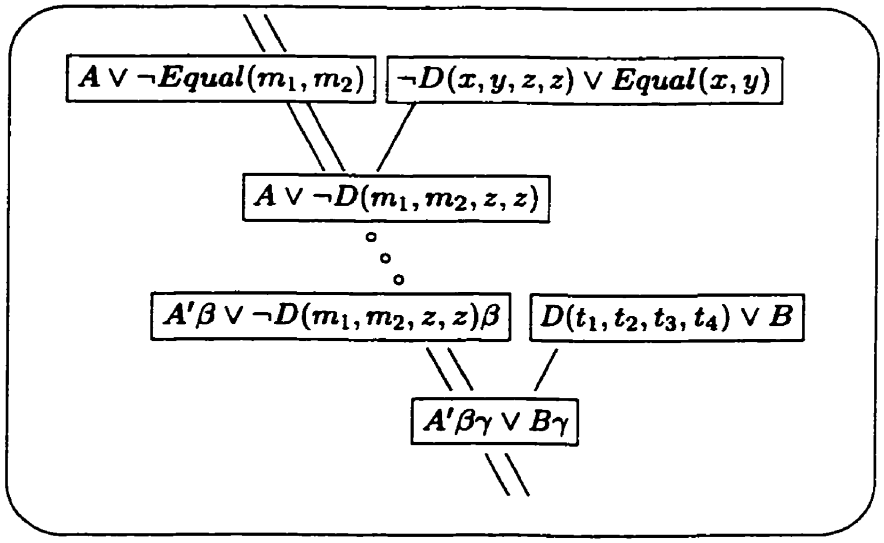
$A \vee \neg Equal(m_1, m_2)$
$\neg D(x, y, z, z) \vee Equal(x, y)$
$A \vee \neg D(m_1, m_2, z, z)$
$A'\beta \vee \neg D(m_1, m_2, z, z)\beta$
$D(t_1, t_2, t_3, t_4) \vee B$
$A'\beta\gamma \vee B\gamma$

Figure 5.5: $P$ before transformation

If that is the case, the structure of the proof $P$ can be rearranged as depicted in Figure 5.6. Since in the original proof $P$, a negative equality literal of the current clause resolves with the equality literal of the equidistance identity axiom, negative equidistance literal will appear in the inference. This literal must eventually resolve with some positive equidistance literal further along the way (*see Figure 5.5*).

It is not hard to see that it is possible to rearrange the order of resolutions so that the resolution with equidistance literal of identity axiom occurs first (*see Figure 5.6*)

By applying this transformation we obtain a resolution which can be easily substituted by equidistance identification and transferred into $P'$ $\square$

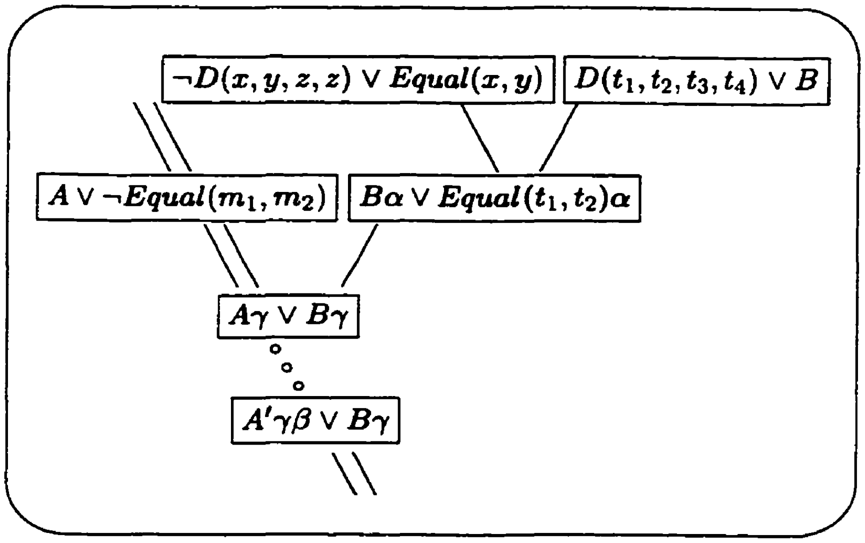
$\neg D(x,y,z,z) \vee Equal(x,y)$
$D(t_1,t_2,t_3,t_4) \vee B$
$A \vee \neg Equal(m_1,m_2)$
$B\alpha \vee Equal(t_1,t_2)\alpha$
$A\gamma \vee B\gamma$
$A'\gamma\beta \vee B\gamma$

Figure 5.6: $P$ after transformation

Equidistance identification considered here can be extended to also accomodate second identity of equidistance axiom:

$$\neg D(x,x,y,z) \vee Equal(y,z)$$

The identity of betweenness was described by the following axiom:

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

Obviously enough, consideration for symmetry of betweenness does not result in another distinct case of identity axiom.

We can introduce a resolution rule of *Betweenness identification* similar to that of equidistance identification.

Thus, if a clause $C$ has a positive betweenness literal: $B(t_1,t_2,t_3)$, and this literal's first and third arguments unify with $\alpha$, i.e. $(t_1\alpha = t_3\alpha)$, infer $C\alpha - B(t_1,t_2,t_3)\alpha + Equal(t1,t2)\alpha$ from $C$, where "-" has a syntactic meaning of "not containing" and "+" has a syntactic meaning of "including".

It can be shown that discarding of betweenness identity axiom while introducing betweenness identification preserves completeness of a proving procedure. The proof of this fact is similar to the one considered for equidistance identification.

The advantages of these inference rules are similar to that of equidistance and betweenness assertion. Additional advantage is that out of two possible ways to resolve with the clauses representing identity axioms only one will remain, thus some irrelevant inferences will not be made.

### 5.2.4 Approaches for symmetries

Both main geometric predicates of Tarskian axiom system as well as an auxillary predicate of colinearity are symmetric. The axioms to describe symmetries are not explicitly present in the axiom system but are easily derivable.

There are eight different congruencies for equidistance, since the order inside both pairs of arguments does not matter as well as the order among the pairs themselves. This potentially induces seven different clauses to describe symmetries of equidistance:

$$\neg D(x,y,u,v) \lor D(y,x,u,v) \qquad \neg D(x,y,u,v) \lor D(x,y,v,u)$$

$$\neg D(x,y,u,v) \lor D(y,x,v,u) \qquad \neg D(x,y,u,v) \lor D(u,v,x,y)$$

$$\neg D(x,y,u,v) \lor D(v,u,x,y) \qquad \neg D(x,y,u,v) \lor D(u,v,y,x)$$

$$\neg D(x,y,u,v) \lor D(v,u,y,x)$$

There is also a symmetry between outer arguments of betweenness:

$$\neg B(x,y,z) \lor B(z,y,x)$$

Since order of arguments does not matter for the colinearity predicate, there is six different congruences and as many as five clauses to describe them:

$$\neg C(x,y,z) \lor C(x,z,y) \qquad \neg C(x,y,z) \lor C(y,x,z)$$

$$\neg C(x,y,z) \vee C(y,z,x) \qquad \neg C(x,y,z) \vee C(z,y,x)$$

$$\neg C(x,y,z) \vee C(z,x,y)$$

It is prohibitively expensive to allow each equidistance literal to appear in eight different forms, colinearity in six and even to allow every betweenness literal to assume two different shapes.

One straightforward approach to help with clause retention using hash tables is to make hashcodes of geometric predicates independent from symmetries.

Assuming that we have a function $hc(t)$ allowing to compute the hashcode for term $t$, the symmetry independent hashcode for a betweenness predicate $B(t_1, t_2, t_3)$ can be computed as function of $(hc(t_1) + hc(t_3))$ and $hc(t_2))$ where $+$ denotes arithmetic addition. Since arithmetic addition is symmetric, the resulting code will be independent of the order of $t_1$ and $t_3$.

Similarly for the predicate of equidistance $D(t_1, t_2, t_3, t_4)$, the hashcode can be computed as a function of $((hc(t_1)+hc(t_2)) \oplus (hc(t_3)+hc(t_4)))$ where $\oplus$ denotes bitwise exclusive-or. Since both addition and bitwise exclusive or are symmetric this will produce the same hashcode for all symmetric equidistances.

Any symmetric function, of course, will suit the above, not just addition or exclusive or.

The hashcode of colinearity predicate $C(t_1, t_2, t_3)$ can be computed simply as a function of $(hc(t_1) + hc(t_2) + hc(t_3))$. Thus, the order of arguments will not matter at all.

Although modifying the retention mechanism to account for symmetries is very important for overall performance, we may also want to modify the inference rules so that the clauses describing symmetries become unnessesary.

Since every equidistance literal has eight different congruences, all describing the same geometric situation, when two equidistance literals are resolved upon there are as many as 64 ways of performing the unification if symmetry is also taken into account. Of these only eight are actually distinct and hence there can be up to eight potentially different resolvents.

If completeness is to be maintained all these must be searched.

Thus, we can modify the resolution and unification algorithms so that when two equidistance or betweenness predicates are attempted for unification symmetries are considered internally.

If completeness is not sought, we can weaken the above and stop when first way of unifying is found and search only one obtained resolvent. From practical experience, this approach appears to be beneficial when proving many elementary geometry theorems.

### 5.2.5 Approaches for transitivities

Both geometric predicates of Tarskian axiom system are transitive. Although a simple enough inference rule with the internal knowledge of transitivity is not obvious, one practical observation should be made.

Many theorems include several versions of equidistance transitivity axiom, some of which involve symmetries.

For instance whereas regular equidistance transitivity is expressed as

$$\neg D(x,y,z,v) \vee \neg D(z,v,u,w) \vee D(x,y,u,w)$$

very often another version will also appear:

$$\neg D(x,y,z,v) \vee \neg D(x,y,u,w) \vee D(z,v,u,w)$$

This is done to minimize the impact of the absence of symmetry axioms, which are derivable from other clauses.

If symmetries are handled by the refinements discussed in the previous section, extra transitivity axioms will not serve any meaningful purpose and should be discarded.

### 5.2.6 Heuristics

Whereas previous sections described refinements which had some observable foundation in the structure of axioms, this section describes heuristics which are based on less precise observations.

Some of the longer axioms of Tarskian axiomatization are very rarely used in the proofs of elementary geometry theorems. For instance, the axiom of elementary continuity is used extremely rarely. This axiom however helps to produce a very considerable number of inferences (or rather does not help in doing so). Thus, one simple heuristic is to discard one or several long clauses from the set.

Constants are very important to geometric reasoning. Thus a possible heuristic is to search grounded clauses only, the ones which do not have any variables. This heuristic does not seem to help in all cases but in a few instances of difficult theorems, it helped to find a proof in a just a few seconds whereas the search considering ungrounded clauses takes many times longer.

As hypotheses of many theorems, clauses representing equalities of some constants are often given. Such equalities can be treated as demodulators to simplify other clauses. Whereas using demodulation for every inferred clause as a post-processing step is relatively expensive, it often helps to demodulate every base clause before the start of the search using demodulators found among other base clauses which will also include equalities of some constants.

The following incomplete heuristic often contributes to faster proofs. The geometric assertion can be applied only to very short clauses, either with a single literal or two literals. This reduces somewhat the number of inferences yet does not seem to prevent discovery of most proofs.

## 5.3 Experiments with implementation

GLIDE (Geometry Linear Iterative Deepening Engine) is a theorem prover for first order predicate calculus. It incorporates refinements aimed at theorems with equality and theorems in Tarskian geometry.

The prover is implemented in C and should compile and run on most UNIX machines. The source code along with sets of test theorems are available from author's web page:

*http://www.cs.mcgill.ca/~savs*

GLIDE recognizes two input formats of theorems, the one supported by TGTP prover[3] and another one used by TPTP (Thousands of Problems for Theorem Provers) library.[4] Two sets of Lex/Yacc based parsers were built and the selected theorem format must be specified at compile time as makefile's parameter.

The run-time options specifying which strategies and refinements to use can be chosen by modifying GLIDE's configuration file.

Several sets of theorems were used in experiments. General strategies and, to some extend, equality refinements were tested using 84 theorems of Stickel's test set. In some settings, GLIDE is capable of solving all of these theorems with the time limit of 900 seconds per theorem.

Geometry refinements were tested on 66 theorems originating in Quaife's work [Qu89] as well as on 165 problems from geometry section of TPTP. The latter set incorporates the former in a somewhat modified form.

The experiments were performed on a LINUX machine with 233Mhz AMD-K6 CPU, 512K L2 cache and 32M of main RAM.

The following sections describe the structure of the prover and the experiments that were done.

[3]TGTP was developed by Professor M. Newborn at McGill University.

[4]TPTP library is maintained by G. Sutcliffe and C. Suttner and it contains sets of theorems from various areas of mathematics.

### 5.3.1 GLIDE theorem prover

The main search strategy implemented in GLIDE uses iteratively deepening depth first search to look for a linear refutation proof. Although in many other implementations, sorted strategies are often preferred to linear strategies, the latter are nonetheless quite attractive and interesting as the search space is explored in a more structured manner and thus performance is more stable compared with sorted strategies which invariably use a version of a best first search in a space where "best" is often poorly definable. Some advantages of sorted strategies are nevertheless exploited in GLIDE by the virtue of using extended search strategy so that the prover looks beyond current maximum iteration depth if it is easy to reduce the number of literals in the current clause. Since the contradiction is a clause of zero length, this heuristic prefers clauses which have a potential to lead to a contradiction.

Use of a hash table to retain unit clauses and of a shallow tree to retain demodulators affects the structure of proofs found which may actually be non-linear but consist of multiple proof lines. One of the proof lines leads to the contradiction and others to unit clauses or demodulators which were used to obtain inferences on other lines.

When and if contradiction is found, the main proof line is printed. If some inferences on this line used resolutions with unit clauses retained by the hash table or were demodulated, the search is restarted to find the lines leading towards these units and demodulators. If these lines themselves used unit clauses from the hash table or demodulators the process is repeated. Beside restoring the proof, verification is also done to insure that no hashing errors occurred during the search.

When a theorem is read, clause retention limits are automatically computed (unless these were preset in the configuration file). These include the limit on number of literals, the limit on number of variables and the limit on number of terms a clause could have to remain searchable.

Further, a simplification routine is called (when appropriate options are set). Basic simplification attempts to delete clauses with pure literals, clauses subsumed by other clauses. It substitutes two clauses with a single resolvent by this resolvent. Similarly, if some resolvent or factor subsumes its parent, the routine substitutes this resolvent or factor for the parent. If equality or geometry refinements will be used, it is possible at this stage to remove all equality and some geometry axioms.

The main loop of the prover controls maximum iterative deepening. If the search ends before time limit expires having explored all inferences at maximum depth, the clause retention limits and maximum iterative depth are increased and the search is restarted.

The routine controlling which clauses from the base set are to be searched is called from the main loop. This routine decides, based on the options, which clauses must be searched. For instance if linear negated conclusion (set of support) refinement is switched on, only clauses from the negated conclusion will be considered for searching at this, first level (*see Section 4.2.1*).

Each clause selected is passed to the searching routine. This routine does the following:

- Checks if a clause is a contradiction. If that's the case, the proof line is printed and additional searches made to find all additional lines leading towards used unit clauses and demodulators.
- Checks if the time limit has been reached. If so, the prover terminates.
- The depth of the clause is compared with the current maximum search depth. If the extended search is allowed, a flag will be set so that to limit possible inferences further on.
- Decides if a clause is worth pursuing. Depending on the options selected the following tests may be made:
  - Hash table is checked to see if this clause has been searched before. Normally, this is only done for clauses in the extended region, but

can optionally be done for every clause.

  - Test is made to see if the current clause is s-subsumed by some base clause or the clause on the line leading towards the current clause.
  - Test is made to see if the current clause is subsumed by some unit base clause or unit clause on the line leading towards the current clause.
  - Alternatively, a test can be made to check if a clause is subsumed ($\theta$-subsumed) by any base clause or any clause on the line leading towards this clause.

  The clause searched before or subsumed will not be searched any further.

- The hash code of the current clause is stored into the hash table.
- If an appropriate option is selected, a test is made to find out if the current clause back s-subsumes some base clause or a clause on the line leading toward the current clause.
- Depending on the options selected, the inferences are made. The following inference rules are available:

  - Binary factoring
  - Binary resolution
  - Identity assertion
  - Paramodulation
  - Geometry assertion
  - Geometry identification

  If geometry refinements are enabled, special symmetry invariant hashing and unification will be used for geometry predicates.

For inference rules requiring two hypotheses, the current clause is used as one and any base clause or any clause on the line leading toward the current clause may be used as the other. Some options may put restriction as to which clause can be used as the other parent. If linear base refinement is enabled, only the base clause can be chosen. If linear merge refinement is enabled, only base clauses or merge clauses on the line towards the current clause can be chosen (*see Section 4.2.1*). If the extended search flag was switched on earlier, further restrictions will be imposed: only those resolution which will reduce the number of the literals in the current clause will be allowed.

- All clauses inferred are post-processed.
  - The clauses whose parameters exceed retention limits are discarded.
  - Equality literals are oriented so that the left side of equality becomes heavier (*see Section 4.3.2*).
  - Simple tautologies such as $L \vee \neg L$, equational tautologies such as $Equal(t,t)$ and geometric tautologies such as $D(t_1, t_2, t_1, t_2)$, $B(t_1, t_1, t_2)$ or $C(t_1, t_1, t_2)$ etc. are detected in the inferred clauses and such clauses are discarded.
  - Simple factorings (such as $L \vee L$ substitued by $L$), simple identity assertions (deleting literals of the type $\neg Equal(t,t)$), and simple geometric assertions (deleting liteals of the type $\neg D(t_1, t_2, t_1, t_2)$, $\neg B(t_1, t_1, t_2)$ or $\neg C(t_1, t_1, t_2)$ etc.) are performed.
  - A test is made to see if any literal of the inferred clause can be resolved with a unit clause stored in the hash table. At this point, it may also be determined that the clause is subsumed by the unit clause in the hash table and thus that it should be discarded. It should be noted that variable numbering local to

each literal is used to obtain literal hashcodes independent from literal's position in the clause.

  - If an appropriate option is selected, all inferred clauses are demodulated.
  - If an appropriate option is selected, a test is made to check if some inferred clause is s-subsumed by another clause inferred at this level.

- Unit clauses are inserted into the hash table along with some variants where variables were substituted by constants. The same hash table is used for both clause hashcodes (for determination if the clause was searched before) and unit clause hashcodes (for unit resolutions). An attribute stored together with every item describes the type of the item, the depth where it was discovered etc. Probing is used for collision resolution. Insertion of unit clauses is prioritized when a collision does occur.

- Unit clauses which were determined to be demodulators are stored in demodulator's shallow tree. This data structure may be considered as the few first levels of the discrimination tree (*see Section 4.4.2*) where first letters of the left term of equality are indexed. Every leaf contains an array of demodulators all sharing the same short prefix. This allows to speed up the demodulation so that only some demodulators are considered when matching a term.

- Clauses are sorted according to their number of literals (Bucketing is used, thus the cost is quite small).

- The inferred clauses are searched recursively starting from the shortest ones.

It should be noted that virtually all steps outlined above can be switched on and off by modifying GLIDE's configuration file. A large number of different search strategies may be obtained by modifying the options.

If a contradiction is found, the proof is printed which may induce additional searches to find how some unit clauses and demodulators were produced. The hashing errors will be reported. Since the hashcodes are 64 bits long, this allows for $2^{64}$ unique codes. Considering the fact that only from 10 to 30 thousand clauses are searched per second with typical search times from 1 to 900 seconds, hashing errors are very unlikely.

Since a lot of different inference steps are taken, the printout of the proof, may be somewhat difficult to read. Appendix A briefly annotates an example of a proof.

### 5.3.2 Experiments

Although multiple sets of theorems were used to test GLIDE, the results on two sets of geometry theorems will be described here. The first set originating in Quaife's article [Qu89] contains 66 geometry theorems, some of which are quite difficult. The second set is taken from the TPTP problem library and contains 165 geometry theorems. Of these only four are in Hilbert geometry whereas the remaining ones are in Tarskian geometry with some using McCharen et al axioms and others Quaife's axioms. The second set includes the first one to a large extend, yet the formulation of theorems often differs.

GLIDE's results for four different strategies will be shown for both sets. Also, the results of *OTTER*[5] and *SPASS*[6] theorem provers will be compared with GLIDE's performance.

Table 5.1 shows the results obtained on Quaife's set. Please note that GLIDE's result with zero resolutions indicates the situation when a unit

[5]OTTER theorem prover was written by William McCune at Argonne National Labs [Mc94].

[6]SPASS theorem prover was developed by Christoph Weidenbach, Bijan Afshordel, Uwe Brahm, Christian Cohrs, Thorsten Engel, Georg Jung, Enno Keen, Christian Theobalt and Dalibor Topic primarily at Max-Planck-Institut für Informatik.

conflict was located during pre-processing before the actual search.

Results for four different strategies are shown in the later tables. For each strategy, first column indicates the iteration at which the proof was found and the time taken to find a contradiction.

The iteration number is not equivalent to the depth of the proof since extended search strategy was used and also due to non-linear nature of the proofs found. It is convenient nonetheless for comparing performance of strategies.

The second column of Table 5.1 gives total number of inferences searched, including during additional searches to restore the proof. The refinements used are indicated above each column.

The experiment depicted in first columns was done using only refined linear strategy. Eight theorems remained unproven after 900 seconds of search time per theorem.

The second experiment additionally used refinements for equality. Six theorems weren't proven. If compared with the first experiment, in 26 cases search with equality refinements actually took much longer whereas only in 12 cases there was any significant improvement (including proving a theorem previously unproven).

The third experiment used both equality and geometry refinements. Four theorems weren't proven. There appears fair improvement compared with the second experiment. This is seen in 35 cases, whereas in 8 cases performance has deteriorated somewhat.

Some quite dramatic effects of the introduced refinements are especially easy to see in the proofs of simple theorems. For example *Q03D1.THM* expresses the theorem that equidistance is symmetric for its first pair of arguments. The proof obtained in the run where geometry refinements were not available was four resolutions deep. A slightly edited listing of this proof is provided below (for annotated example of actual GLIDE listing please refer to AppendixA).

```
 1 D(a,b,b,a)
 2 ~D(a,b,c,d) | ~D(a,b,e,f) | D(c,d,e,f)
19 ~D(a,b,c,d) | D(c,d,a,b)
20 D(a(),b(),c(),d())
21 ~D(b(),a(),c(),d()) NC

 22(21a,19b) ~D(c(),d(),b(),a()) NC
  23(22a,2c) ~D(a,b,c(),d()) | ~D(a,b,b(),a()) NC
   24(23b,1a) ~D(a(),b(),c(),d()) NC
    25(24a,20a) #
```

In the listing above, base clauses used in the proof are given first followed by the proof line where for each clause the way of producing it is indicated. For instance, the clause 22 was obtained by resolving the first literal (letter *a* indicates the first literal) of the clause 21 with the second literal (letter *b*) of the clause 19.

To derive the symmetry of first argument pair of equidistance, the proof used already available symmetry between argument pairs of equidistance (clause 19) as well as transitivity (clause 2) and reflexivity (clause 1) of equidistance.

Since one of the geometry refinements explicitly makes the resolution of equidistance predicates invariant with respect to all symmetries, the proof of the same theorem when the geometry refinements were available is only a single resolution deep.

```
15 D(a(),b(),c(),d())
16 ~D(b(),a(),c(),d()) NC

 17 (16a,15a) #
```

It is much more difficult to precisely see the effects of geometry refinements when analyzing more complex proofs. The proofs may look significantly different and it may not be possible to, for instance, see the spots where several resolutions were replaced by an application of a new resolution rule. As there are always very large numbers of different proofs for the same theorem, even a minor change of a search strategy is likely to cause finding a distinctly different proof. However, the general observation is that the proofs involving geometry refinements practically always (and as expected) involve less steps. The proof may often be found on an earlier iteration. Moreover, since many clauses describing the axioms can be discarded before the search which uses geometry refinements, such a search often requires less inferences. The latter fact can be observed in the results of the experiments.

The usefulness of the refinements can be also verified by examining how often these were used in the found proofs. Table 5.2 lists the resolution rules which were used in the proofs of the theorems of Quaife's set when the geometry and equality refinements were available. Please note that DISTANCE, BETWEEN and COLINEAR appearing in that table indicate the use of resolution rules of either geometry assertion or identification for respective predicates whereas UNIF_D, UNIF_B and UNIF_C indicate the use of symmetry invariant unification in binary resolution of respective geometric predicates. As it can be seen, the refinements were used in almost 70% of all proofs. This percentage is even higher if we consider only difficult theorems.

Finally, the last experiment used some equality refinements, notably orientation of equality literal, identity assertion (but not paramodulation) combined with all geometry refinements. Only three theorems were not solved. This strategy is the best if compared with all the other ones. It defeats the first strategy in 48 cases and only looses in 2. It similarly defeats strategies two and three.

One of the reasons why the last strategy appears to be the best is due to the way the theorems in this set are formulated. All substituitivity axioms are missing, yet many other axioms appear in several different forms. This is one reason why paramodulation performed so poorly. Since paramodulation models substitutivity, it was doing the work weaker strategies did not even have to do explicitly. Second reason is that paramodulation tends to produce inferences earlier in the search tree thus increasing the fan-out. Although the resulting proofs will be shorter, it still may be necessary to look at a larger number of inferences in order to find a proof.

Table 5.3 shows results obtained by running SPASS 1.0 and OTTER 3.0 provers in their automated modes and running GLIDE using the best geometry strategy.

It must be mentioned that Quaife's article [Qu89] describes proving all theorems from this set using an earlier version of OTTER. However, some of it was done by using different strategies for different theorems. Performance of GLIDE can also be improved by using different strategies for different theorems.

When given 900 seconds per theorem and running in automated mode SPASS was unable to prove 20 of the 66 theorems whereas OTTER failed in 18 cases. GLIDE, when using geometry refinements, cannot find a proof for only three theorems. It can be argued that both SPASS and OTTER were operating in automated mode and that by fine-tuning the strategies a better performance can be obtained. One of the purposes of this comparison, however, is to demonstrate that proving some of the geometry theorems by using best available provers may not be completely obvious or easy and that specialized strategies for geometry may be necessary.

It is very difficult to compare details of the performance statistics of two different provers. For instance, comparing the depth of the obtained proofs and total number of resolutions performed may not be meaningful due to significant differences of search strategies. For instance GLIDE does considerably more inferences per second then either of the two other provers. The complexity of the inferences, however, is different, and so are the rules determining which inferences to count. Both OTTER and SPASS immediately discard many inferences without counting them as searched.

When comparing rough measures of performance: the time required to find a proof and the total number of proofs found, GLIDE with its geometry strategies appears to be doing reasonably well.

Table 5.1: GLIDE's results on Quaife's set.

| Name | d/t | | d/t | eq | d/t | eq geo | d/t | low eq geo |
|---|---|---|---|---|---|---|---|---|
| Q01D1.THM | 1/0 | 7 | 1/0 | 23 | 1/0 | 16 | 1/1 | 8 |
| Q02D2.THM | 1/0 | 25 | 1/0 | 33 | 1/0 | 0 | 1/0 | 0 |
| Q03D3.THM | 1/0 | 21 | 1/0 | 33 | 1/0 | 0 | 1/0 | 0 |
| Q04D4.THM | 1/0 | 1032 | 1/1 | 1605 | 1/0 | 0 | 1/0 | 0 |
| Q05D5.THM | 1/0 | 394 | 1/1 | 283 | 1/0 | 26 | 1/0 | 22 |
| Q06E1.THM | 1/0 | 23 | 1/1 | 47 | 1/0 | 1713 | 1/1 | 608 |
| Q07B0.THM | 1/0 | 386 | 1/1 | 3576 | 1/1 | 845 | 1/0 | 177 |
| Q08R2.THM | 1/0 | 1573 | 1/0 | 4440 | 1/1 | 4750 | 1/1 | 30 |
| Q09R3.THM | 1/1 | 4893 | 1/1 | 4082 | 2/2 | 25232 | 1/1 | 694 |
| Q10R4.THM | 1/1 | 27381 | 1/2 | 29109 | 1/2 | 18675 | 1/1 | 3131 |
| Q11D7.THM | 1/1 | 285 | 1/1 | 1005 | 1/0 | 30 | 1/0 | 21 |
| Q12D8.THM | 2/7 | 274357 | 2/13 | 383822 | 2/31 | 884083 | 2/2 | 25308 |
| Q13D9.THM | 1/1 | 260 | 1/1 | 221 | 1/2 | 69575 | 1/2 | 8342 |
| Q14D10A.THM | 1/0 | 421 | 1/0 | 3287 | 1/0 | 484 | 1/1 | 271 |
| Q14D10B.THM | 1/13 | 288916 | 1/15 | 299657 | 1/30 | 552352 | 1/3 | 33819 |
| Q14D10C.THM | 2/12 | 504248 | 2/11 | 300454 | 1/2 | 53354 | 2/7 | 109253 |
| Q15R5.THM | 1/5 | 116796 | 1/6 | 110798 | 1/1 | 574 | 1/3 | 29757 |
| Q16R6.THM | 2/10 | 475421 | 2/7 | 204103 | 1/1 | 1217 | 1/3 | 32949 |
| Q17T3.THM | 1/1 | 16 | 1/1 | 42 | 1/1 | 27 | 1/1 | 5 |
| Q18B1.THM | 1/1 | 5285 | 1/1 | 1613 | 1/1 | 8351 | 1/1 | 1940 |
| Q19T1.THM | 2/9 | 376317 | 1/4 | 107777 | 1/1 | 0 | 1/0 | 0 |
| Q20T2.THM | 1/1 | 85 | 1/1 | 104 | 1/1 | 28 | 1/1 | 6 |
| Q21B2.THM | -/- | - | 2/792 | 18259304 | 2/196 | 4243999 | 1/4 | 61896 |
| Q22B3.THM | 1/5 | 96831 | 1/5 | 129928 | 1/0 | 1044 | 1/1 | 397 |
| Q23T6.THM | 1/4 | 92629 | 1/6 | 128797 | 1/4 | 64230 | 1/2 | 8336 |
| Q24B4.THM | 2/13 | 307407 | 1/3 | 53680 | 1/2 | 33249 | 1/3 | 42297 |
| Q25B5.THM | 2/14 | 343353 | 2/11 | 315282 | 1/1 | 3747 | 2/3 | 70311 |
| Q26B6.THM | 1/6 | 137498 | 1/7 | 192839 | 1/5 | 98065 | 1/2 | 13991 |
| Q27B7.THM | 2/24 | 732509 | 1/11 | 329064 | 1/1 | 1764 | 1/1 | 490 |
| Q28B8.THM | 1/1 | 5428 | 1/1 | 12273 | 1/1 | 6151 | 1/1 | 3660 |
| Q29B9.THM | 1/1 | 6879 | 1/1 | 13006 | 1/2 | 6975 | 1/1 | 4053 |
| Q30E2.THM | 1/2 | 34961 | 1/1 | 1463 | 1/1 | 1630 | 1/2 | 13476 |
| Q31E3.THM | -/- | - | 2/179 | 2451144 | 1/1 | 302 | 1/2 | 6864 |
| Q32B10.THM | 1/50 | 779460 | -/- | - | 1/1 | 1037 | 1/1 | 394 |
| Q33D11.THM | 1/8 | 149631 | 1/23 | 512642 | 1/31 | 641205 | 1/4 | 54494 |
| Q34D12.THM | -/- | - | -/- | - | -/- | - | -/- | - |
| Q35D13.THM | 1/2 | 14402 | 1/2 | 13543 | 1/1 | 1877 | 1/1 | 507 |
| Q36D14.THM | 1/10 | 208069 | 1/21 | 458232 | 1/38 | 577708 | 1/8 | 72307 |
| Q37D15.THM | 1/7 | 133593 | 1/19 | 404258 | 1/25 | 469382 | 1/3 | 44492 |
| Q38I2A.THM | 1/2 | 7918 | 2/247 | 4637558 | 1/7 | 73165 | 1/2 | 6118 |
| Q38I2B.THM | -/- | - | -/- | - | -/- | - | 3/558 | 4726011 |
| Q38I2C.THM | 1/8 | 126541 | 1/20 | 364351 | 1/23 | 256750 | 1/7 | 46861 |
| Q39I3.THM | 1/5 | 50415 | 1/48 | 873797 | 1/60 | 996328 | 1/3 | 18158 |
| Q40I4.THM | -/- | - | 1/132 | 2224129 | 1/274 | 3841853 | 1/659 | 1848224 |
| Q41B11.THM | 3/172 | 3637913 | 2/535 | 11906002 | 1/31 | 546983 | 1/2 | 11826 |
| Q42B12.THM | -/- | - | -/- | - | -/- | - | -/- | - |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Q43B13.THM | 1/4 | 120770 | 1/17 | 630461 | 1/24 | 437930 | 1/3 | 46446 |
| Q44T7.THM | 2/115 | 2677383 | 1/47 | 1807250 | 1/3 | 33351 | 1/6 | 82670 |
| Q45T9.THM | 1/76 | 1728321 | 1/129 | 3695868 | 1/16 | 387766 | 1/4 | 21083 |
| Q46B14.THM | 1/2 | 1831 | 1/1 | 4127 | 1/2 | 5587 | 1/2 | 1120 |
| Q47T8.THM | 2/119 | 2813371 | 1/46 | 1370541 | 1/83 | 1471267 | 1/6 | 49426 |
| Q48B15.THM | 1/1 | 171 | 1/1 | 172 | 1/2 | 3680 | 1/2 | 3948 |
| Q49C2.THM | 2/124 | 3280571 | 1/54 | 2009342 | 1/1 | 1982 | 1/1 | 1799 |
| Q50T10.THM | -/- | - | -/- | - | 1/2 | 362 | 1/2 | 209 |
| Q51T11.THM | 1/1 | 18 | 1/2 | 81 | 1/2 | 120 | 1/2 | 1 |
| Q52C3.THM | 1/1 | 1641 | 1/11 | 170614 | 1/2 | 0 | 1/2 | 0 |
| Q53C4.THM | 1/14 | 375381 | 1/50 | 3489724 | 1/54 | 961348 | 1/2 | 3600 |
| Q54T12.THM | 2/125 | 3217479 | 1/66 | 3917873 | 1/18 | 310461 | 1/5 | 41232 |
| Q55C5.THM | 2/134 | 8249844 | 2/337 | 20196479 | 1/58 | 2076422 | 1/5 | 51377 |
| Q56T13.THM | 1/14 | 296366 | 1/24 | 568236 | 1/37 | 673146 | 1/3 | 16397 |
| Q57W1A.THM | 1/80 | 5255363 | 1/183 | 18433142 | 1/6 | 104416 | 1/11 | 208418 |
| Q57W1B.THM | 1/54 | 3179806 | 1/159 | 11581025 | 1/6 | 208080 | 1/24 | 471562 |
| Q57W1C.THM | 1/55 | 2203318 | 1/135 | 8953915 | 1/90 | 4094155 | 1/18 | 370248 |
| Q58W2A.THM | 1/18 | 397767 | 1/64 | 1292125 | 1/17 | 369095 | 1/7 | 59232 |
| Q58W2B.THM | 1/22 | 1051096 | 1/60 | 1219633 | 1/63 | 1791712 | 1/7 | 74572 |
| Q59W3.THM | -/- | - | -/- | - | -/- | - | -/- | - |

Table 5.2: Resolution rules used to prove theorems of Quaife's set.

| Name | Rules |
|---|---|
| Q01D1.THM | DISTANCE |
| Q02D2.THM | UNIF_D |
| Q03D3.THM | UNIF_D |
| Q04D4.THM | BINRES UNIF_D |
| Q05D5.THM | BINRES |
| Q06E1.THM | BINRES FACTOR DISTANCE BETWEEN |
| Q07B0.THM | PARAMOD |
| Q08R2.THM | BINRES PARAMOD |
| Q09R3.THM | PARAMOD DISTANCE |
| Q10R4.THM | BINRES FACTOR PARAMOD DISTANCE BETWEEN UNIF_D |
| Q11D7.THM | BINRES DISTANCE |
| Q12D8.THM | BINRES PARAMOD UNIF_D |
| Q13D9.THM | BINRES FACTOR DISTANCE BETWEEN UNIF_D |
| Q14D10A.THM | BINRES PARAMOD |
| Q14D10B.THM | BINRES |
| Q14D10C.THM | BINRES ASSERT DISTANCE DEMOD UNIF_D UNIF_B |
| Q15R5.THM | BINRES DISTANCE BETWEEN |
| Q16R6.THM | BINRES PARAMOD UNIF_B |
| Q17T3.THM | BETWEEN |
| Q18B1.THM | BINRES PARAMOD BETWEEN DEMOD |
| Q19T1.THM | UNIF_B |
| Q20T2.THM | BETWEEN |
| Q21B2.THM | BINRES PARAMOD BETWEEN |
| Q22B3.THM | BINRES UNIF_B |
| Q23T6.THM | BINRES UNIF_B |
| Q24B4.THM | BINRES PARAMOD UNIF_B |
| Q25B5.THM | BINRES PARAMOD BETWEEN UNIF_B |
| Q26B6.THM | BINRES |
| Q27B7.THM | BINRES PARAMOD UNIF_B |
| Q28B8.THM | BINRES PARAMOD |
| Q29B9.THM | BINRES PARAMOD |
| Q30E2.THM | BINRES PARAMOD BETWEEN |
| Q31E3.THM | BINRES PARAMOD DISTANCE |
| Q32B10.THM | BINRES UNIF_B |
| Q33D11.THM | BINRES FACTOR PARAMOD BETWEEN UNIF_D |
| Q35D13.THM | BINRES FACTOR |
| Q36D14.THM | BINRES DISTANCE UNIF_D |

| | |
|---|---|
| Q37D15.THM | BINRES FACTOR PARAMOD BETWEEN UNIF_D |
| Q38I2A.THM | BINRES PARAMOD |
| Q38I2C.THM | BINRES FACTOR |
| Q39I3.THM | BINRES PARAMOD |
| Q40I4.THM | BINRES PARAMOD |
| Q41B11.THM | BINRES DISTANCE BETWEEN UNIF_D |
| Q43B13.THM | BINRES PARAMOD BETWEEN UNIF_B |
| Q44T7.THM | BINRES PARAMOD UNIF_B |
| Q45T9.THM | BINRES PARAMOD UNIF_B |
| Q46B14.THM | BINRES |
| Q47T8.THM | BINRES |
| Q48B15.THM | BINRES FACTOR UNIF_D |
| Q49C2.THM | BINRES UNIF_B UNIF_C |
| Q50T10.THM | BINRES FACTOR UNIF_B UNIF_C |
| Q51T11.THM | BINRES PARAMOD |
| Q52C3.THM | BINRES DEMOD |
| Q53C4.THM | BINRES |
| Q54T12.THM | BINRES FACTOR BETWEEN UNIF_B |
| Q55C5.THM | BINRES PARAMOD UNIF_C |
| Q56T13.THM | BINRES PARAMOD UNIF_C |
| Q57W1A.THM | BINRES DISTANCE BETWEEN UNIF_D UNIF_B |
| Q57W1B.THM | BINRES DISTANCE BETWEEN UNIF_D UNIF_B |
| Q57W1C.THM | BINRES DISTANCE BETWEEN UNIF_B |
| Q58W2A.THM | BINRES FACTOR PARAMOD DISTANCE BETWEEN DEMOD UNIF_D |
| Q58W2B.THM | BINRES FACTOR PARAMOD DISTANCE BETWEEN DEMOD UNIF_D |

Table 5.3: Comparison of SPASS, OTTER and GLIDE on Quaife's set.

| Name | d/t | SPASS | d/t | OTTER | d/t | GLIDE |
|---|---|---|---|---|---|---|
| Q01D1.THM | 3/0 | 2 | 2/0 | 55 | 1/1 | 8 |
| Q02D2.THM | 3/0 | 18 | 1/0 | 85 | 1/0 | 0 |
| Q03D3.THM | 3/0 | 23 | 1/1 | 88 | 1/0 | 0 |
| Q04D4.THM | 4/0 | 27 | 6/2 | 1716 | 1/0 | 0 |
| Q05D5.THM | 4/0 | 224 | 2/3 | 1786 | 1/0 | 22 |
| Q06E1.THM | 2/0 | 10 | 1/0 | 268 | 1/1 | 608 |
| Q07B0.THM | 2/0 | 7 | 1/0 | 364 | 1/0 | 177 |
| Q08R2.THM | 3/0 | 14 | 1/0 | 45 | 1/1 | 30 |
| Q09R3.THM | 5/0 | 9 | 4/0 | 45 | 1/1 | 694 |
| Q10R4.THM | 4/0 | 40 | 6/2 | 1999 | 1/1 | 3131 |
| Q11D7.THM | 2/0 | 32 | 2/0 | 427 | 1/0 | 21 |
| Q12D8.THM | -/- | - | 19/697 | 66540 | 2/2 | 25308 |
| Q13D9.THM | -/- | - | 2/5 | 4377 | 1/2 | 8342 |
| Q14D10A.THM | 4/477 | 9901 | 0/1 | 1033 | 1/1 | 271 |
| Q14D10B.THM | -/- | - | -/- | - | 1/3 | 33819 |
| Q14D10C.THM | -/- | - | -/- | - | 2/7 | 109253 |
| Q15R5.THM | 4/0 | 408 | -/- | - | 1/3 | 29757 |
| Q16R6.THM | -/- | - | -/- | - | 1/3 | 32949 |
| Q17T3.THM | 2/0 | 4 | 1/1 | 161 | 1/1 | 5 |
| Q18B1.THM | 4/0 | 21 | 4/0 | 171 | 1/1 | 1940 |
| Q19T1.THM | 7/200 | 8976 | 4/187 | 18696 | 1/0 | 0 |
| .Q20T2.THM | 2/0 | 6 | 1/0 | 82 | 1/1 | 6 |
| Q21B2.THM | 7/58 | 5682 | 6/200 | 33563 | 1/4 | 61896 |
| Q22B3.THM | 3/0 | 79 | 3/1 | 743 | 1/1 | 397 |
| Q23T6.THM | 4/0 | 439 | 2/5 | 4146 | 1/2 | 8336 |
| Q24B4.THM | 8/179 | 7845 | 6/234 | 35374 | 1/3 | 42297 |
| Q25B5.THM | 4/0 | 600 | 4/1 | 981 | 2/3 | 70311 |
| Q26B6.THM | -/- | - | 3/11 | 5379 | 1/2 | 13991 |
| Q27B7.THM | 7/1 | 1079 | 4/2 | 1650 | 1/1 | 490 |
| Q28B8.THM | 5/7 | 1915 | 3/1 | 985 | 1/1 | 3660 |
| Q29B9.THM | 4/0 | 622 | 3/1 | 1003 | 1/1 | 4053 |
| Q30E2.THM | 4/0 | 25 | 22/14 | 5529 | 1/2 | 13476 |
| Q31E3.THM | -/- | - | -/- | - | 1/2 | 6864 |
| Q32B10.THM | 4/1 | 908 | -/- | - | 1/1 | 394 |
| Q33D11.THM | -/- | - | 4/24 | 12084 | 1/4 | 54494 |
| Q34D12.THM | -/- | - | -/- | - | -/- | - |
| Q35D13.THM | -/- | - | 2/3 | 5784 | 1/1 | 507 |
| Q36D14.THM | -/- | - | 2/13 | 8634 | 1/8 | 72307 |
| Q37D15.THM | -/- | - | 6/21 | 14668 | 1/3 | 44492 |
| Q38I2A.THM | 4/0 | 630 | 3/1 | 5070 | 1/2 | 6118 |
| Q38I2B.THM | -/- | - | -/- | - | 3/558 | 4726011 |
| Q38I2C.THM | 5/592 | 15584 | -/- | - | 1/7 | 46861 |
| Q39I3.THM | 5/122 | 7646 | 4/75 | 17353 | 1/3 | 18158 |
| Q40I4.THM | 5/147 | 6282 | -/- | - | 1/659 | 1848224 |
| Q41B11.THM | -/- | - | -/- | - | 1/2 | 11826 |
| Q42B12.THM | -/- | - | -/- | - | -/- | - |

| | | | | | | |
|---|---|---|---|---|---|---|
| Q43B13.THM | 4/53 | 4198 | 3/55 | 16434 | 1/3 | 46446 |
| Q44T7.THM | -/- | - | 5/204 | 36192 | 1/6 | 82670 |
| Q45T9.THM | 9/67 | 4257 | 4/855 | 116321 | 1/4 | 21083 |
| Q46B14.THM | 7/63 | 4322 | 0/1 | 1030 | 1/2 | 1120 |
| Q47T8.THM | 6/62 | 4187 | -/- | - | 1/6 | 49426 |
| Q48B15.THM | 4/10 | 2206 | 3/4 | 7572 | 1/2 | 3948 |
| Q49C2.THM | 5/0 | 407 | -/- | - | 1/1 | 1799 |
| Q50T10.THM | 5/58 | 4200 | -/- | - | 1/2 | 209 |
| Q51T11.THM | 2/0 | 6 | 0/1 | 514 | 1/2 | 1 |
| Q52C3.THM | 4/0 | 36 | 4/0 | 778 | 1/2 | 0 |
| Q53C4.THM | 6/532 | 13871 | -/- | - | 1/2 | 3600 |
| Q54T12.THM | 8/90 | 6074 | -/- | - | 1/5 | 41232 |
| Q55C5.THM | 10/68 | 5252 | 6/26 | 11004 | 1/5 | 51377 |
| Q56T13.THM | 8/100 | 7329 | 5/26 | 10368 | 1/3 | 16397 |
| Q57W1A.THM | -/- | - | 6/140 | 29207 | 1/11 | 208418 |
| Q57W1B.THM | -/- | - | 9/224 | 45063 | 1/24 | 471562 |
| Q57W1C.THM | -/- | - | 6/232 | 45500 | 1/18 | 370248 |
| Q58W2A.THM | 5/0 | 483 | 9/20 | 12883 | 1/7 | 59232 |
| Q58W2B.THM | 7/0 | 317 | 8/29 | 12976 | 1/7 | 74572 |
| Q59W3.THM | -/- | - | -/- | - | -/- | - |

Table 5.4 shows results obtained by GLIDE on TPTP's geometry theorems.

The comparative performance of strategies is similar to that shown on Quaife's set. The weakest strategy failed to find a proof in roughly half (80 out of 165) of all cases. When equality refinements are available only three more proofs are additionally found and the performance worsens or improves in roughly the same number of cases. Since all axioms of substitutivity of equality are available in this set, introduction of paramodulation has a better impact here compared with Quaife's set.

When geometry refinements become available, a fair performance improvement can be observed. Instead of failing on half of theorems, no proof is obtained in one third of the cases (54 out of 165). We can also note that many theorems were proven faster than before. Only in a few instances it is not so, with the most dramatic example perhaps being *GEO077-4.p*. This example, however, only validates the general rule since this is one of rare theorems in Hilbert geometry and thus our geometry refinements were quite useless.

The strategy where geometry refinements are employed but some equality refinements (i.e. paramodulation) are not available allows to find the proof in three more cases leaving the number of unproven theorems at 51. This strategy again appears to be the best, however, the performance difference with the strategy where paramodulation was used appears to be less severe than in Quaife's set. This is probably due to the absence of equality substituitivity axioms in the latter set.

Table 5.5 shows the results obtained by running SPASS 1.0 and OTTER 3.0 provers in their automated modes and running GLIDE using the best geometry strategy.

The comparative performance is similar to that seen on Quaife's set. In this case, both SPASS and OTTER outperform GLIDE when the latter does not use geometry refinements, yet show weaker performance compared with GLIDE's refined strategy. Thus, SPASS cannot prove 64 out of 165 theorems whereas OTTER fails in 78 out of 165 cases. Without the refinements GLIDE fails in 80 out of 165 cases, however, with the refinements the number of unproven theorems drops to 51.

Table 5.4: GLIDE's results on TPTP's set.

| Name | d/t | | d/t | eq | d/t | eq geo | d/t | low eq geo |
|---|---|---|---|---|---|---|---|---|
| GEO001-1.p | -/- | - | -/- | - | 1/0 | 0 | 1/0 | 0 |
| GEO001-2.p | -/- | - | -/- | - | 1/0 | 0 | 1/0 | 0 |
| GEO001-3.p | -/- | - | -/- | - | 1/0 | 0 | 1/1 | 0 |
| GEO001-4.p | -/- | - | -/- | - | 1/0 | 4 | 1/1 | 4 |
| GEO002-1.p | -/- | - | -/- | - | 1/1 | 12 | 1/0 | 8 |
| GEO002-2.p | -/- | - | -/- | - | 1/1 | 6 | 1/0 | 5 |
| GEO002-3.p | 1/1 | 21 | 1/1 | 28 | 1/1 | 20 | 1/0 | 7 |
| GEO002-4.p | 2/1 | 8248 | 2/1 | 8248 | 1/0 | 8 | 1/0 | 4 |
| GEO003-1.p | 2/2 | 16905 | 2/3 | 36551 | 1/0 | 10 | 1/1 | 6 |
| GEO003-2.p | 2/1 | 8803 | 2/1 | 8287 | 1/0 | 6 | 1/1 | 5 |
| GEO003-3.p | 1/1 | 16 | 1/1 | 28 | 1/0 | 19 | 1/1 | 6 |
| GEO004-1.p | 3/540 | 17996750 | 2/13 | 284355 | 1/3 | 64932 | 2/16 | 269465 |
| GEO004-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO005-1.p | -/- | - | 2/17 | 327884 | 1/5 | 170570 | 2/21 | 348959 |
| GEO005-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO006-1.p | -/- | - | -/- | - | 1/1 | 14014 | 1/1 | 4508 |
| GEO006-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO006-3.p | 1/1 | 604 | 1/1 | 233 | 1/1 | 176 | 1/0 | 748 |
| GEO007-1.p | -/- | - | -/- | - | 1/1 | 10974 | 1/1 | 8185 |
| GEO007-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO007-3.p | 2/70 | 1977254 | 1/17 | 656494 | 1/4 | 109949 | 1/15 | 163260 |
| GEO008-1.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO008-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO008-3.p | 2/87 | 2691530 | 1/47 | 1218867 | 1/23 | 694563 | 1/3 | 19412 |
| GEO009-1.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO009-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO009-3.p | 2/83 | 2222369 | 1/26 | 1422754 | 1/4 | 52165 | 1/5 | 69378 |
| GEO010-1.p | -/- | - | -/- | - | 1/1 | 0 | 1/0 | 0 |
| GEO010-2.p | -/- | - | -/- | - | 1/1 | 0 | 1/0 | 0 |
| GEO010-3.p | -/- | - | -/- | - | 1/2 | 0 | 1/1 | 0 |
| GEO011-1.p | -/- | - | -/- | - | 1/0 | 7 | 1/1 | 13 |
| GEO011-2.p | 1/0 | 31 | 1/1 | 22 | 1/0 | 4 | 1/1 | 17 |
| GEO011-3.p | 1/1 | 39 | 1/2 | 72 | 1/1 | 78 | 1/2 | 60 |
| GEO011-4.p | 1/0 | 32 | 1/1 | 22 | 1/0 | 19 | 1/0 | 23 |
| GEO011-5.p | 1/0 | 31 | 1/0 | 19 | 1/0 | 16 | 1/0 | 22 |
| GEO012-1.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO012-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO012-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO013-1.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO013-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO013-3.p | 1/4 | 36079 | 1/49 | 2216204 | 1/12 | 449859 | 1/2 | 7451 |
| GEO014-2.p | 1/0 | 7 | 1/0 | 23 | 1/0 | 16 | 1/1 | 8 |
| GEO015-2.p | 1/1 | 38 | 1/0 | 46 | 1/0 | 0 | 1/0 | 0 |
| GEO015-3.p | 1/1 | 25 | 1/0 | 33 | 1/0 | 0 | 1/1 | 0 |
| GEO016-2.p | 1/1 | 23 | 1/1 | 31 | 1/0 | 0 | 1/1 | 0 |
| GEO016-3.p | 1/0 | 21 | 1/1 | 33 | 1/0 | 0 | 1/1 | 0 |
| GEO017-2.p | 1/0 | 810 | 1/1 | 987 | 1/1 | 0 | 1/1 | 0 |
| GEO017-3.p | 1/0 | 133 | 1/0 | 105 | 1/1 | 0 | 1/0 | 0 |
| GEO018-2.p | 1/0 | 36 | 1/0 | 44 | 1/1 | 0 | 1/0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| GEO018-3.p | 1/1 | 167 | 1/0 | 139 | 1/1 | 0 | 1/0 | 0 |
| GEO019-2.p | 1/1 | 23 | 1/0 | 31 | 1/1 | 0 | 1/0 | 0 |
| GEO019-3.p | 1/1 | 16 | 1/1 | 28 | 1/0 | 0 | 1/0 | 0 |
| GEO020-2.p | 1/0 | 780 | 1/1 | 959 | 1/0 | 0 | 1/1 | 0 |
| GEO020-3.p | 1/0 | 63 | 1/1 | 57 | 1/0 | 0 | 1/1 | 0 |
| GEO021-2.p | 1/0 | 63 | 1/0 | 71 | 1/0 | 0 | 1/1 | 0 |
| GEO021-3.p | 1/1 | 73 | 1/0 | 67 | 1/0 | 0 | 1/1 | 0 |
| GEO022-2.p | 1/1 | 959 | 1/1 | 1090 | 1/0 | 26 | 1/0 | 22 |
| GEO022-3.p | 1/1 | 394 | 1/1 | 283 | 1/1 | 26 | 1/0 | 22 |
| GEO024-2.p | 1/0 | 2319 | 1/1 | 1511 | 1/1 | 24 | 1/1 | 26 |
| GEO024-3.p | 1/1 | 285 | 1/0 | 1005 | 1/1 | 30 | 1/1 | 21 |
| GEO025-2.p | 3/33 | 1457792 | 2/12 | 416996 | 2/22 | 808358 | 2/2 | 16212 |
| GEO025-3.p | 2/10 | 361113 | 2/14 | 438149 | 2/7 | 163247 | 2/3 | 46135 |
| GEO026-2.p | 2/3 | 74368 | 2/2 | 58970 | 1/1 | 36035 | 1/1 | 3052 |
| GEO026-3.p | 1/1 | 1485 | 1/0 | 221 | 1/2 | 69575 | 1/2 | 8342 |
| GEO027-2.p | 2/2 | 50803 | 2/2 | 43467 | 1/2 | 26055 | 1/1 | 2119 |
| GEO027-3.p | 1/0 | 43 | 1/1 | 3287 | 1/1 | 484 | 1/1 | 283 |
| GEO028-2.p | -/- | - | -/- | - | 2/206 | 4963288 | 2/7 | 130536 |
| GEO028-3.p | 1/7 | 120709 | 1/13 | 271638 | 1/31 | 522089 | 1/3 | 30366 |
| GEO029-2.p | -/- | - | -/- | - | 1/6 | 234867 | 1/0 | 2241 |
| GEO029-3.p | 2/21 | 646730 | 1/14 | 312374 | 1/1 | 2961 | 1/1 | 1745 |
| GEO030-2.p | 1/2 | 45672 | 1/2 | 43433 | 1/2 | 20109 | 1/1 | 5110 |
| GEO030-3.p | 1/8 | 149750 | 1/23 | 494585 | 1/30 | 628523 | 1/4 | 54220 |
| GEO031-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO031-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO032-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO032-3.p | 1/2 | 14402 | 1/2 | 13543 | 1/1 | 1877 | 1/1 | 507 |
| GEO033-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO033-3.p | 1/11 | 211123 | 1/22 | 459385 | 1/39 | 585595 | 1/7 | 73141 |
| GEO034-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO034-3.p | 1/11 | 200055 | 1/21 | 449707 | 1/25 | 471919 | 1/4 | 45908 |
| GEO035-2.p | 1/0 | 143 | 1/0 | 39 | 1/1 | 1724 | 1/0 | 623 |
| GEO035-3.p | 1/0 | 398 | 1/0 | 47 | 1/1 | 1713 | 1/0 | 612 |
| GEO036-2.p | -/- | - | -/- | - | 2/1 | 2364 | 2/0 | 2170 |
| GEO036-3.p | 1/1 | 12429 | 1/1 | 540 | 1/1 | 303 | 1/1 | 5112 |
| GEO037-2.p | -/- | - | -/- | - | 2/180 | 3981797 | 2/30 | 558999 |
| GEO037-3.p | -/- | - | 2/137 | 3357675 | 1/1 | 302 | 1/2 | 9408 |
| GEO038-2.p | 1/0 | 373 | 1/1 | 734 | 1/1 | 1024 | 1/0 | 212 |
| GEO038-3.p | 1/1 | 463 | 1/1 | 3550 | 1/1 | 1027 | 1/0 | 229 |
| GEO039-2.p | 2/1 | 13857 | 2/1 | 6815 | 2/0 | 4065 | 1/1 | 567 |
| GEO039-3.p | 1/1 | 2793 | 1/1 | 712 | 1/1 | 5616 | 1/1 | 2223 |
| GEO040-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO040-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO041-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO041-3.p | 1/2 | 42813 | 1/2 | 47492 | 1/1 | 483 | 1/1 | 237 |
| GEO042-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO042-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO043-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO043-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO044-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO044-3.p | -/- | - | -/- | - | -/- | - | 3/253 | 5250862 |
| GEO045-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO045-3.p | 2/12 | 444547 | 1/3 | 99099 | 1/2 | 12668 | 1/1 | 1816 |
| GEO046-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO046-3.p | 2/25 | 650574 | 2/192 | 5902932 | 1/2 | 19035 | 2/5 | 136946 |
| GEO047-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO047-3.p | 1/1 | 3663 | 1/1 | 5607 | 1/1 | 2722 | 1/1 | 1791 |
| GEO048-2.p | -/- | - | -/- | - | 1/1 | 55 | 1/0 | 11 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| GEO048-3.p | 1/3 | 22473 | 1/217 | 3718431 | 1/2 | 1037 | 1/1 | 394 |
| GEO049-2.p | -/- | - | 3/390 | 9975075 | 3/198 | 8735042 | 2/1 | 11674 |
| GEO049-3.p | 1/194 | 6462964 | 1/217 | 8396135 | 1/60 | 1375085 | 1/8 | 64535 |
| GEO050-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO050-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO051-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO051-3.p | 1/3 | 58085 | 1/7 | 128644 | 1/6 | 139835 | 1/2 | 8317 |
| GEO052-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO052-3.p | 1/3 | 36683 | 1/4 | 119395 | 1/4 | 49673 | 1/3 | 12414 |
| GEO053-2.p | -/- | - | -/- | - | 2/46 | 1776190 | 3/283 | 7265791 |
| GEO053-3.p | 1/2 | 3117 | 1/2 | 172 | 1/2 | 3670 | 1/2 | 3946 |
| GEO054-2.p | 1/0 | 269 | 1/0 | 671 | 1/0 | 497 | 1/0 | 134 |
| GEO054-3.p | 1/0 | 7 | 1/1 | 29 | 1/0 | 19 | 1/1 | 7 |
| GEO055-2.p | 1/0 | 303 | 1/1 | 1272 | 1/1 | 2504 | 1/1 | 122 |
| GEO055-3.p | 1/1 | 619 | 1/2 | 16134 | 1/2 | 13390 | 1/1 | 93 |
| GEO056-2.p | 1/1 | 2517 | 1/0 | 67 | 1/0 | 1995 | 1/0 | 748 |
| GEO056-3.p | 1/1 | 379 | 1/1 | 22 | 1/1 | 1130 | 1/1 | 16 |
| GEO057-2.p | 1/0 | 1394 | 1/1 | 72 | 1/1 | 1955 | 1/1 | 920 |
| GEO057-3.p | 1/1 | 357 | 1/1 | 22 | 1/1 | 1130 | 1/0 | 15 |
| GEO058-2.p | 1/1 | 4148 | 2/3 | 30599 | 2/6 | 127958 | 1/0 | 1036 |
| GEO058-3.p | 1/2 | 28272 | 1/2 | 29811 | 1/2 | 18675 | 1/1 | 3302 |
| GEO059-2.p | 3/47 | 1409378 | -/- | - | -/- | - | 3/37 | 652867 |
| GEO059-3.p | 1/7 | 128572 | 1/6 | 110798 | 1/1 | 574 | 1/3 | 32381 |
| GEO060-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO061-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO061-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO062-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO062-3.p | 1/7 | 88512 | 1/46 | 1492618 | 1/90 | 1425068 | 1/19 | 305599 |
| GEO063-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO063-3.p | -/- | - | -/- | - | -/- | - | 1/43 | 480011 |
| GEO064-2.p | -/- | - | -/- | - | 1/0 | 6 | 1/0 | 6 |
| GEO064-3.p | 1/1 | 37 | 1/2 | 85 | 1/2 | 51 | 1/2 | 6 |
| GEO065-2.p | -/- | - | -/- | - | 1/1 | 13 | 1/1 | 16 |
| GEO065-3.p | 1/2 | 183 | 1/2 | 206 | 1/2 | 181 | 1/2 | 68 |
| GEO066-2.p | -/- | - | -/- | - | 1/0 | 20 | 1/0 | 26 |
| GEO066-3.p | 1/2 | 318 | 1/2 | 323 | 1/1 | 306 | 1/1 | 125 |
| GEO067-2.p | 3/136 | 8670661 | 3/4 | 213130 | 1/0 | 0 | 1/0 | 0 |
| GEO067-3.p | 1/1 | 2382 | 1/5 | 74335 | 1/2 | 0 | 1/2 | 0 |
| GEO068-2.p | -/- | - | -/- | - | 3/336 | 26451437 | 2/5 | 168954 |
| GEO068-3.p | 1/70 | 3080605 | 1/147 | 10235134 | 1/54 | 948038 | 1/3 | 5820 |
| GEO069-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO069-3.p | 2/57 | 3562602 | 2/146 | 10051613 | 1/18 | 662438 | 1/2 | 8220 |
| GEO070-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO070-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO071-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO071-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO072-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO072-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO073-1.p | -/- | - | 2/110 | 2277026 | 1/11 | 434553 | 2/80 | 1532115 |
| GEO073-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO073-3.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO074-2.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO075-2.p | -/- | - | -/- | - | 1/1 | 16 | 1/0 | 8 |
| GEO076-4.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO077-4.p | 1/1 | 17424 | 1/202 | 3673531 | 1/204 | 3673531 | 1/1 | 12318 |
| GEO078-4.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO078-5.p | -/- | - | -/- | - | -/- | - | -/- | - |
| GEO079-1.p | 0/0 | 0 | 0/0 | 0 | 0/0 | 0 | 0/0 | 0 |

Table 5.5: Comparison of SPASS, OTTER and GLIDE on TPTP's set.

| Name | d/t | SPASS | d/t | OTTER | d/t | GLIDE |
|---|---|---|---|---|---|---|
| GEO001-1.p | 7/0 | 135 | 6/840 | 34541 | 1/0 | 0 |
| GEO001-2.p | 6/0 | 111 | -/- | - | 1/0 | 0 |
| GEO001-3.p | 6/6 | 2085 | 4/189 | 17872 | 1/1 | 0 |
| GEO001-4.p | -/- | - | 4/249 | 208798 | 1/1 | 4 |
| GEO002-1.p | 6/1 | 531 | 6/187 | 19552 | 1/0 | 8 |
| GEO002-2.p | 6/1 | 404 | 6/227 | 19671 | 1/0 | 5 |
| GEO002-3.p | 2/0 | 6 | 1/0 | 57 | 1/0 | 7 |
| GEO002-4.p | 11/1 | 405 | 8/1 | 259 | 1/0 | 4 |
| GEO003-1.p | 3/0 | 11 | 2/0 | 577 | 1/1 | 6 |
| GEO003-2.p | 3/0 | 10 | 2/0 | 556 | 1/1 | 5 |
| GEO003-3.p | 2/0 | 4 | 1/0 | 161 | 1/1 | 6 |
| GEO004-1.p | -/- | - | 13/270 | 63149 | 2/16 | 269465 |
| GEO004-2.p | -/- | - | -/- | - | -/- | - |
| GEO005-1.p | -/- | - | 18/269 | 63124 | 2/21 | 348959 |
| GEO005-2.p | -/- | - | -/- | - | -/- | - |
| GEO006-1.p | 12/3 | 665 | -/- | - | 1/1 | 4508 |
| GEO006-2.p | 9/1 | 386 | -/- | - | -/- | - |
| GEO006-3.p | 4/0 | 73 | 2/5 | 4308 | 1/0 | 748 |
| GEO007-1.p | 12/20 | 2059 | -/- | - | 1/1 | 8185 |
| GEO007-2.p | -/- | - | -/- | - | -/- | - |
| GEO007-3.p | -/- | - | 5/71 | 16274 | 1/15 | 163260 |
| GEO008-1.p | -/- | - | -/- | - | -/- | - |
| GEO008-2.p | -/- | - | -/- | - | -/- | - |
| GEO008-3.p | 9/21 | 2539 | -/- | - | 1/3 | 19412 |
| GEO009-1.p | -/- | - | -/- | - | -/- | - |
| GEO009-2.p | -/- | - | -/- | - | -/- | - |
| GEO009-3.p | 5/101 | 5808 | -/- | - | 1/5 | 69378 |
| GEO010-1.p | 15/2 | 691 | -/- | - | 1/0 | 0 |
| GEO010-2.p | 10/1 | 477 | -/- | - | 1/0 | 0 |
| GEO010-3.p | 5/0 | 413 | -/- | - | 1/1 | 0 |
| GEO011-1.p | 7/1 | 581 | -/- | - | 1/1 | 18 |
| GEO011-2.p | 2/0 | 1 | 0/0 | 55 | 1/1 | 17 |
| GEO011-3.p | 2/0 | 6 | 0/1 | 391 | 1/2 | 60 |
| GEO011-4.p | 2/0 | 1 | 0/0 | 55 | 1/0 | 23 |
| GEO011-5.p | 2/0 | 1 | 0/0 | 34 | 1/0 | 22 |
| GEO012-1.p | -/- | - | -/- | - | -/- | - |
| GEO012-2.p | -/- | - | -/- | - | -/- | - |
| GEO012-3.p | -/- | - | -/- | - | -/- | - |
| GEO013-1.p | -/- | - | -/- | - | -/- | - |
| GEO013-2.p | -/- | - | -/- | - | -/- | - |
| GEO013-3.p | 12/151 | 12468 | 5/25 | 5642 | 1/2 | 7451 |
| GEO014-2.p | 3/0 | 2 | 2/0 | 55 | 1/1 | 8 |
| GEO015-2.p | 4/0 | 54 | 3/0 | 274 | 1/0 | 0 |
| GEO015-3.p | 3/0 | 22 | 1/0 | 87 | 1/1 | 0 |
| GEO016-2.p | 3/0 | 22 | 1/0 | 63 | 1/1 | 0 |
| GEO016-3.p | 3/0 | 24 | 1/0 | 88 | 1/1 | 0 |
| GEO017-2.p | 5/0 | 58 | 4/1 | 807 | 1/1 | 0 |
| GEO017-3.p | 3/0 | 43 | 2/0 | 619 | 1/0 | 0 |
| GEO018-2.p | 4/0 | 57 | 2/1 | 610 | 1/0 | 0 |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| GEO018-3.p | 4/0 | 62 | 2/1 | 818 | 1/0 | 0 |
| GEO019-2.p | 3/0 | 22 | 1/0 | 65 | 1/0 | 0 |
| GEO019-3.p | 2/0 | 12 | 1/0 | 94 | 1/0 | 0 |
| GEO020-2.p | 5/0 | 61 | 4/1 | 803 | 1/1 | 0 |
| GEO020-3.p | 2/0 | 12 | 2/0 | 612 | 1/1 | 0 |
| GEO021-2.p | 6/0 | 64 | 2/1 | 607 | 1/1 | 0 |
| GEO021-3.p | 3/0 | 50 | 2/1 | 809 | 1/1 | 0 |
| GEO022-2.p | 6/0 | 101 | 4/2 | 1703 | 1/0 | 22 |
| GEO022-3.p | 3/0 | 65 | 2/3 | 1786 | 1/0 | 22 |
| GEO024-2.p | 3/0 | 9 | 2/1 | 555 | 1/1 | 26 |
| GEO024-3.p | 2/0 | 32 | 2/0 | 427 | 1/1 | 21 |
| GEO025-2.p | 9/333 | 7441 | 22/687 | 39710 | 2/2 | 16212 |
| GEO025-3.p | -/- | - | 19/753 | 74545 | 2/3 | 46135 |
| GEO026-2.p | 7/260 | 7070 | 4/2 | 1800 | 1/1 | 3052 |
| GEO026-3.p | -/- | - | 2/5 | 3662 | 1/2 | 8342 |
| GEO027-2.p | -/- | - | 6/435 | 20055 | 1/1 | 2119 |
| GEO027-3.p | 4/0 | 70 | 0/0 | 1033 | 1/1 | 283 |
| GEO028-2.p | -/- | - | -/- | - | 2/7 | 130536 |
| GEO028-3.p | -/- | - | -/- | - | 1/3 | 30366 |
| GEO029-2.p | -/- | - | -/- | - | 1/0 | 2241 |
| GEO029-3.p | -/- | - | -/- | - | 1/1 | 1745 |
| GEO030-2.p | 10/493 | 12113 | 6/23 | 5758 | 1/1 | 5110 |
| GEO030-3.p | -/- | - | 4/19 | 10110 | 1/4 | 54220 |
| GEO031-2.p | -/- | - | -/- | - | -/- | - |
| GEO031-3.p | -/- | - | -/- | - | -/- | - |
| GEO032-2.p | -/- | - | -/- | - | -/- | - |
| GEO032-3.p | -/- | - | 2/3 | 4856 | 1/1 | 507 |
| GEO033-2.p | -/- | - | -/- | - | -/- | - |
| GEO033-3.p | 4/0 | 548 | 2/12 | 8396 | 1/7 | 73141 |
| GEO034-2.p | -/- | - | -/- | - | -/- | - |
| GEO034-3.p | -/- | - | 6/19 | 14430 | 1/4 | 45908 |
| GEO035-2.p | 2/0 | 9 | 1/1 | 259 | 1/0 | 623 |
| GEO035-3.p | 2/0 | 10 | 1/0 | 268 | 1/0 | 612 |
| GEO036-2.p | 5/0 | 20 | 16/59 | 7609 | 2/0 | 2170 |
| GEO036-3.p | 4/0 | 25 | 22/13 | 5531 | 1/1 | 5112 |
| GEO037-2.p | 12/0 | 304 | -/- | - | 2/30 | 558999 |
| GEO037-3.p | -/- | - | -/- | - | 1/2 | 9408 |
| GEO038-2.p | 2/0 | 5 | 1/1 | 497 | 1/0 | 212 |
| GEO038-3.p | 2/0 | 7 | 1/0 | 364 | 1/0 | 229 |
| GEO039-2.p | 4/0 | 30 | 6/5 | 2166 | 1/1 | 567 |
| GEO039-3.p | 4/0 | 29 | 4/0 | 132 | 1/1 | 2223 |
| GEO040-2.p | 5/0 | 100 | 7/514 | 22679 | -/- | - |
| GEO040-3.p | 5/0 | 161 | 6/232 | 33651 | -/- | - |
| GEO041-2.p | 9/6 | 1355 | -/- | - | -/- | - |
| GEO041-3.p | 3/0 | 229 | 3/1 | 660 | 1/1 | 237 |
| GEO042-2.p | 7/4 | 1182 | -/- | - | -/- | - |
| GEO042-3.p | 5/0 | 170 | 6/246 | 34477 | -/- | - |
| GEO043-2.p | 9/4 | 1042 | -/- | - | -/- | - |
| GEO043-3.p | 8/472 | 12836 | 7/331 | 48451 | -/- | - |
| GEO044-2.p | -/- | - | -/- | - | -/- | - |
| GEO044-3.p | -/- | - | 7/131 | 19394 | 3/253 | 5250862 |
| GEO045-2.p | -/- | - | -/- | - | -/- | - |
| GEO045-3.p | 7/9 | 2486 | 4/2 | 1462 | 1/1 | 1816 |
| GEO046-2.p | -/- | - | -/- | - | -/- | - |
| GEO046-3.p | 11/0 | 358 | 6/1 | 1236 | 2/5 | 136946 |
| GEO047-2.p | -/- | - | -/- | - | -/- | - |
| GEO047-3.p | 5/0 | 252 | 3/1 | 851 | 1/1 | 1791 |
| GEO048-2.p | 8/0 | 306 | -/- | - | 1/0 | 11 |

| | | | | | | |
|---|---|---|---|---|---|---|
| GEO048-3.p | 4/18 | 3122 | -/- | - | 1/1 | 394 |
| GEO049-2.p | -/- | - | -/- | - | 2/1 | 11674 |
| GEO049-3.p | -/- | - | -/- | - | 1/8 | 64535 |
| GEO050-2.p | -/- | - | -/- | - | -/- | - |
| GEO050-3.p | -/- | - | -/- | - | -/- | - |
| GEO051-2.p | -/- | - | -/- | - | -/- | - |
| GEO051-3.p | 4/0 | 292 | 3/28 | 10992 | 1/2 | 8317 |
| GEO052-2.p | -/- | - | -/- | - | -/- | - |
| GEO052-3.p | 3/0 | 286 | 3/20 | 10100 | 1/3 | 12414 |
| GEO053-2.p | -/- | - | -/- | - | 3/283 | 7265791 |
| GEO053-3.p | 8/1 | 787 | 3/3 | 5713 | 1/2 | 3946 |
| GEO054-2.p | 2/0 | 5 | 2/0 | 51 | 1/0 | 134 |
| GEO054-3.p | 2/0 | 4 | 2/0 | 45 | 1/1 | 7 |
| GEO055-2.p | 2/0 | 7 | 2/0 | 51 | 1/1 | 122 |
| GEO055-3.p | 2/0 | 23 | 2/0 | 45 | 1/1 | 93 |
| GEO056-2.p | 3/0 | 17 | 3/4 | 1558 | 1/0 | 748 |
| GEO056-3.p | 2/0 | 14 | 2/0 | 45 | 1/1 | 16 |
| GEO057-2.p | 3/0 | 13 | 3/1 | 479 | 1/1 | 920 |
| GEO057-3.p | 2/0 | 13 | 2/0 | 45 | 1/0 | 15 |
| GEO058-2.p | 6/0 | 50 | 9/5 | 2024 | 1/0 | 1036 |
| GEO058-3.p | 4/0 | 41 | 7/2 | 1999 | 1/1 | 3302 |
| GEO059-2.p | 6/0 | 79 | -/- | - | 3/37 | 652867 |
| GEO059-3.p | 4/0 | 408 | -/- | - | 1/3 | 32381 |
| GEO060-2.p | -/- | - | -/- | - | -/- | - |
| GEO061-2.p | -/- | - | -/- | - | -/- | - |
| GEO061-3.p | -/- | - | -/- | - | -/- | - |
| GEO062-2.p | -/- | - | -/- | - | -/- | - |
| GEO062-3.p | 8/416 | 15239 | 6/267 | 31031 | 1/19 | 305599 |
| GEO063-2.p | -/- | - | -/- | - | -/- | - |
| GEO063-3.p | 5/142 | 6546 | -/- | - | 1/43 | 480011 |
| GEO064-2.p | 6/0 | 165 | -/- | - | 1/0 | 6 |
| GEO064-3.p | 2/0 | 72 | 2/1 | 1009 | 1/2 | 6 |
| GEO065-2.p | 6/0 | 165 | -/- | - | 1/1 | 16 |
| GEO065-3.p | 2/0 | 72 | 2/1 | 1008 | 1/2 | 68 |
| GEO066-2.p | 6/0 | 165 | -/- | - | 1/0 | 26 |
| GEO066-3.p | 2/0 | 72 | 2/1 | 1007 | 1/1 | 125 |
| GEO067-2.p | 6/0 | 17 | 7/0 | 801 | 1/0 | 0 |
| GEO067-3.p | 4/0 | 36 | 4/1 | 602 | 1/2 | 0 |
| GEO068-2.p | -/- | - | -/- | - | 2/5 | 168954 |
| GEO068-3.p | 5/0 | 543 | -/- | - | 1/3 | 5820 |
| GEO069-2.p | -/- | - | -/- | - | -/- | - |
| GEO069-3.p | 8/5 | 2189 | 6/24 | 7709 | 1/2 | 8220 |
| GEO070-2.p | -/- | - | -/- | - | -/- | - |
| GEO070-3.p | -/- | - | -/- | - | -/- | - |
| GEO071-2.p | -/- | - | -/- | - | -/- | - |
| GEO071-3.p | -/- | - | -/- | - | -/- | - |
| GEO072-2.p | -/- | - | -/- | - | -/- | - |
| GEO072-3.p | -/- | - | -/- | - | -/- | - |
| GEO073-1.p | -/- | - | -/- | - | 2/80 | 1532115 |
| GEO073-2.p | -/- | - | -/- | - | -/- | - |
| GEO073-3.p | -/- | - | -/- | - | -/- | - |
| GEO074-2.p | -/- | - | -/- | - | -/- | - |
| GEO075-2.p | -/- | - | -/- | - | 1/0 | 8 |
| GEO076-4.p | 18/0 | 374 | -/- | - | -/- | - |
| GEO077-4.p | 10/9 | 2775 | -/- | - | 1/1 | 12318 |
| GEO078-4.p | -/- | - | -/- | - | -/- | - |
| GEO078-5.p | -/- | - | -/- | - | -/- | - |
| GEO079-1.p | 2/0 | 4 | 2/0 | 2 | 5/0 | 0 |

# Chapter 6

# Conclusions

Simple refinement for theorems in Tarskian-Euclidian geometry which were outlined in Chapter 5 do seem to help to improve efficiency of the resolution based provers.

The experiments which were carried out show that the number of unproven theorems is reduced by a significant percentage (from 30% to 50%) and the theorems which were proven before usually take less time to be solved when geometry refinements are available.

It should be noted that the refinements proposed require only minor knowledge of the axiom system. They can also be applied in domains with similar types of axioms. After all, properties of reflexivity, symmetry and identity are very common. Thus, these refinements are quite generic and even the decision whether they should be employed for a particular theorem can be easily automated. It is indeed not hard to discover identity or reflexivity axioms and apply special resolution rules to predicates which possess these properties.

The experiments performed also allow us to make the following quite interesting observations:

- Using paramodulation may slow down significantly a prover which is using a linear search strategy. Since paramodulation tends to infer in one step what would have taken several binary resolutions, many inferences appear earlier in the tree, considerably increasing its fan-out. Although paramodulation potentially shortens the path towards the contradiction, it may still increase the number of clauses we have to look at before it is found.

- Inexpensive versions of subsumption, particularely combination of s-subsumption and unit subsumpition often account for as many as 80 to 90% of all subsumptions. Hence, introduction of a very expensive $\theta$-subsumption may not bring considerable gain.
- Back s-subsumption does not seem to help in the greater majority of situations.
- Use of variable ordering local to every literal to produce hashcodes independent from literal's position in the clause does help, but not to the degree hoped.
- Demodulation of all base clauses before the search by demodulators found among the base clauses seems to help significantly in a number of instances.
- Although geometry identification inference rule does not reduce the number of literals in the current clause, it seems to help when this rule is used in the extended region.
- Symmetry invariant unification appears to be very useful. Although for complex predicates there may be several distinct ways of symmetry invariant unification for two literals, pursuing only one and discarding the rest does not seem to worsen the search strategy.

As possible future extension of this work it could be interesting, beside finding refinements based on the properties of predicates, to also consider what properties of geometric functions may be used to build refinements.

It is also interesting to try some sorted strategies together with geometry refinements as well as to experiment with discrimination trees used for retention of unit clauses and to compare its performance against that of the hash table.

Since constants are often of great importance for geometric reasoning it is interesting to combine few levels of semantic splitting with the use of resolution in the leafs.

As a final remark, it must be mentioned that automated theorem proving does have multiple applications in the areas such as expert system, hardware and software verification, logical databases etc. Efficient techniques to prove theorems could translate into superior application programs in many different fields.

# Bibliography

[AL91] O.L. Astrachan, D.W. Loveland: *METEOR: High performance theorem provers using model elimination.* in *Automated Reasoning.* R.S. Boyer editor, Kluwer Academic Press (1991)

[AS92] O.L. Astrachan, M. Stickel: *Caching and lemmazing in model elimination theorem proving.* CADE-11, Springer-Verlag (1992)

[BB91] E.J. Borowski, J.M. Borwein: *Dictionary of Mathematics.* Harper (1991)

[Be92] D. Benanav: *Recognising Unnecessary Clauses in Resolution Based Systems.* Journal of Automated Reasoning 9(1):43-76 (1992)

[Bu98] S.N. Burris: *Logic for mathematics and computer science.* Prentice Hall (1998)

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

[Ch88] S.C. Chou: *Mechanical Geometry Theorem Proving.* D. Reidel (1988)

[He73] H. Hermes: *Introduction to mathematical logic.* Springer-Verlag (1973)

[HS96] J.D. Horton, B. Spencer: *Clause trees: A tool for understanding and implementing resolution in automated reasoning.* Technical report, University of New Brunswick (1996)

[Le97] A. Leitsch: *The resolution calculus.* Springer-Verlag (1997)

[Lo78] D.W. Loveland: *Automated theorem proving: A logical basis.* North-Holland (1978)

[MO76] J.D. McCharen, R.A Overbeek, L.A. Wos: *Problems and Experiments for and with Automated Theorem Proving Programs.* IEEE Transactions on Computers C-25(8):773-782 (1976)

[Mc93] W. McCune: *Experiments with discrimination-tree indexing and path indexing for term retrieval.* Journal of Automated Reasoning 9:147-167 (1993)

[Mc94] W. McCune: *Otter 3.0 reference manual and guide.* Argonne National Laboratory (1994)

[Mc97] W. McCune: *33 Basic test problems: A practical evaluation of some paramodulation strategies.* in *Automated reasoning and its applications.* MIT Press (1997)

[MP96] W. McCune, R. Padmanabhan: *Automated Deduction in Equational Logic and Cubic Curves.* Springer Verlag (1996)

[Me64] E. Mendelson: *Introduction to mathematical logic.* D. Van Nostrand (1964)

[Ne97] M. Newborn: *The great theorem prover.* Newborn Software (1997)

[Ni95] H. de Nivelle: *Ordering refinements of resolution.* PhD thesis, Delft University of Technology (1995)

[Qu89] A. Quaife: *Automated Development of Tarski's Geometry.* Journal of Automated Reasoning 5(1):97-118 (1989)

[Ro65] J.A. Robinson: *A Machine-oriented logic based on resolution principle* Journal of the ACM 12:23-41 1965

[Sa98] S. Savchenko: *Theorem proving and database querying.* Dr. Dobbs Journal, 8 (1998)

[Sh67] J.R. Shoenfield: *Mathematical logic.* Addison-Wesley (1967)

[SJ97] R. Socher-Ambrosius, P.Johann: *Duduction Systems.* Springer-Verlag (1997)

[St88] M. Stickel: *A Prolog technology theorem prover: implementation by an extended Prolog compiler.* Journal of Automated Reasoning 4:353-380 (1988)

[Ta98] T. Tammet: *Towards efficient subsumption.* CADE-15, pp. 427-441, Springer-Verlag (1998)

# Appendix A

# Examples of GLIDE's proofs

These proofs were obtained for theorems of Quaife's set using both equality and geometry refinements.

## A.1 Annotated proof of Q58W2A.THM

The following shows an example of GLIDE's proof. The theorem comes from Quaife's set [Qu89] and can be considered as relatively difficult. Some brief annotations will be given below (line numbers on the left were inserted to facilitate the annotation).

```
001   GLIDE r7 Aug 1999 for (Q58W2A.THM)
002
003   P( equal D B C )
004   F( Ext ip p p1 p2 euc1 euc2 cont R ins a b c d e )
005   C( p p1 p2 a b c d e )
006   O( N(Q58W2A.THM) T900 D6/9 SB SEH SGH LN(30) LM LF LX EA PI PF DB GA GI
007      HU HE SFS SFU SCS L7 V16 T9 )
008   S( LO PO B3 FO RO 02 C(-3,2) )
009   S( LO PO BO FO RO 00 C(0,0) )
010   E( 122 112 )
011   G( 112 89 )
012
013   # L( D1 T18 R(120243,15745,48,6518) U(10578,15248,87629)
014        C(76130-0,98905) S(15059,0,0,0) G(106147,14118) V14601 )
```

```

Proof:
------
B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
B35 ~Bxyz ~Byzu Bxzu equalyz BF
B85 Baec
 [D0] BDM85(85a) Baea [~equalac]
B88 ~equalda NC
L89(88a<-10c) ~equalxa ~Bxdy ~Bzdu Bxueuc2xzduy NC BF
 L90(89a,35d) ~Bxdy ~Bzdu Bxueuc2xzduy ~Bvax ~Baxv Bvxv NC BF
  LF90(90b) ~Bxdy Bxyeuc2xxdyy ~Bzax ~Baxu Bzxu NC
   L92(90d,85a) ~Bedx Bexeuc2eedxx ~Byae Byea NC
    [H1] LB93(92c) Baea <~Baea> NC
     [H2] LB94(93a) # <equalae> NC

B84 equalac
R0 equalac

B85 Baec
[D0] P85 Baec
RDM1(85a) Baea [~equalac]

B30 ~Bxyz ~equalxz Buyz
B31 ~Bxyz ~Byxz equalxy BF
B86 Bbed
B88 ~equalda NC
L89(88a<-31c) ~equalxa ~Bxdy ~Bdxy NC BF
 L90(89b,30c) ~equalxa ~Bdxy ~Bzdy ~equalzy NC BF
  LB91(90c) ~equalxa ~Bdxy ~equaldy NC BF
   [H3] R2(91b,86a) ~equalea <~equaldb> NC BF

# L( D1 T23 R(150763,20641,96,7175) U(17846,19468,113240)
     C(97979-0,125265) S(19666,0,0,0) G(139214,18818) V14639 )
B35 ~Bxyz ~Byzu Bxzu equalyz BF
B47 ~Bxyz ~Buvw ~Dxyuv ~Dxzuw Dyzvw BF
B81 Dabcd
 [D0] BDM81(81a) Dabad [~equalac]
B88 ~equalda NC
L89(88a,35d) ~Bxad ~Bady Bxdy NC
 L90(89c,47a) ~Bxad ~Bady ~Bzuv ~Dxdzu ~Dxyzv Ddyuv NC BF
  LD91(90d) ~Bxad ~Bady ~Bdxz ~Dxydz Ddyxz NC BF
   LF91(91c) ~Badx ~Daxda Ddxaa NC
    [H4] L93(91b,81a) Ddbaa <~Ddbaa> NC
```

```
     RD3(93a) equaldb NC

# L( D1 T25 R(164136,22923,144,7361) U(23232,20984,124967)
     C(107496-0,136391) S(21806,0,0,0) G(154122,20409) V14763 )
B83 Dacbd
[D0] P83 Dacbd
RDM4(83a) Daabd [~equalac]

Resolution rules used:
----------------------
BINRES FACTOR PARAMOD DISTANCE BETWEEN DEMOD UNIF_D

Used clauses:
------------
 B0 BxyExtxyzu
 B1 DxExtyzzuzu
 B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dviwj BF
 B3 ~Bxyz ~Buvz Byipxyzvuu BF
 B4 ~Bxyz ~Buvz Bvipxyzvux BF
 B5 ~Bpp1p2
 B6 ~Bp1p2p
 B7 ~Bp2pp1
 B8 ~Dxyxz ~Duyuz ~Dvyvz equalyz Bxuv Buvx Bvxu BF
 B9 ~Bxyz ~Buyv equalxy Bxueuc1xuyvz BF
>B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
 B11 ~Bxyz ~Buyv equalxy Beuc1xuyvzzeuc2xuyvz BF
 B12 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Bzcontxyzwuvv BF
 B13(0a,1a) ~Dxyxz ~Dxuxv ~Bxyu ~Bywu ~equalcontxyzwuvi Dxwxi BF
 B14 ~Dxyzu ~Dzuvw Dxyvw BF
 B15 equalxExtyxzz
 B16 ~equalxExtyzuv Byzx
 B17 equalRxyExtxyxy
 B18 BxyRxy
 B19 DxRyxyx
 B20 ~equalxy equalyRxy
 B21 equalxRxx
 B22 ~equalxRyx equalyx
 B23 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
 B24 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
 B25 ~Bxyz equalxy equalzExtxyyz
 B26 ~Dxyzu equalExtvwxyExtvwzu equalvw
 B27 equalExtxyxyExtxyyx equalxy
 B28 DxyxRRyxx
```

```
101     B29 equalxRRxyy
102    >B30 ~Bxyz ~equalxz Buyz
103    >B31 ~Bxyz ~Byxz equalxy BF
104     B32 ~Bxyz ~Bxzy equalyz BF
105     B33 ~Bxyz ~Byuz Bxyu BF
106     B34 ~Bxyz ~Bxzu Byzu BF
107    >B35 ~Bxyz ~Byzu Bxzu equalyz BF
108     B36 ~Bxyz ~Byzu Bxyu equalyz BF
109     B37 ~Bxyz ~Byuz Bxuz BF
110     B38 ~Bxyz ~Bxzu Bxyu BF
111     B39 ~equalpp1
112     B40 ~equalp1p2
113     B41 ~equalpp2
114     B42 ~equalxExtyxpp1
115     B43 DxExtyxpp1zExtuzpp1
116     B44 ~Bxyz ~Buvz ~Bxwu Bwipvipxwuvzxyzz Byipvipxwuvzxyzv BF
117     B45 ~Bxyz ~Dxzxu ~Dyzyu equalxy equalzu BF
118     B46 ~Dxyzu ~Dxvzw ~Dxizj ~Dviwj ~Bxyv ~Bzuw Dyiuj BF
119    >B47 ~Bxyz ~Buvw ~Dxyuv ~Dxzuw Dyzvw BF
120     B48 ~Dxyzu ~Dyvuw ~Dxizj ~Dviwj ~Bxyv ~Bzuw Dyiuj BF
121     B49 ~Bxyz ~Dxyxu ~Dzyzu equalyu PF
122     B50 equalinsxyzuExtExtyxpp1xzu
123     B51 Dxyzinszuxy
124     B52 ~Bxyz ~Dxzuv Buinsuvxyv
125     B53 ~Bxyz ~Dxzuv Dyzinsuvxyv
126     B54 ~Bxyz equalyinsxzxy
127     B55 ~Dxyzu equalinsvwxyinsvwzu
128     B56 ~Dxyzu ~Dyvuw ~Dxvzw ~Bxyv Bzuw BF
129     B57 ~Bxyz ~Bxyu equalxy Bxzu Bxuz BF
130     B58 ~Bxyz ~Bxyu equalxy Byzu Byuz BF
131     B59 ~Bxyz ~Buyz equalyz Bxuy Buxy BF
132     B60 ~Bxyz ~Bxuz Bxyu Bxuy BF
133     B61 ~Bxyz ~Bxuz Byuz Buyz BF
134     B62 ~Bxyz ~Byuv ~Bxvz Bxuz BF
135     B63 ~Bxyz ~Dxyxz equalyz
136     B64 ~Bxyz Cxyz
137     B65 ~Bxyz Czxy
138     B66 ~Bxyz Cyzx
139     B67 ~Cxyz Bxyz Byzx Bzxy BF
140     B68 ~Bxyz Czyx
141     B69 ~Bxyz Cxzy
142     B70 ~Bxyz Cyxz
143     B71 ~Cpp1p2
```

```
144     B72(2a,3a) ~Dxxyz ~Dxuzv ~Dxuyv Cyzv BF
145     B73 ~equalxy Cxzy
146     B74 ~Dxyzu ~Dyvuw ~Dxvzw ~Cxyv Czuw BF
147     B75 ~Cxyz ~Cxyu Cxzu equalxy BF
148     B76 ~Cxyz ~Cxyu Cyzu equalxy BF
149     B77 ~Cxyz ~Cuyz Cuxz equalzy BF
150     B78 ~Cxyz ~Cuyz Cuxy equalzy BF
151     B79 ~Cxyz ~Cxyu ~Cxyv Czuv equalxy BF
152     B80 ~Dxyzu ~Dyzux ~Dxzyu ~Cxyz ~Bxvz ~Byvu equalxy equalux BF
153    >B81 Dabcd
154     B82 Dbcda
155    >B83 Dacbd
156    >B84 equalac
157    >B85 Baec
158    >B86 Bbed
159     B87 ~equalab NC
160    >B88 ~equalda NC
```

In the above proof, lines 3-5 show statistics of the predicates, functions and constants employed in the theorem. Lines 6-7 show which options are enabled. Further, lines 8-11 display the output of simplification routines. Lines 13-14 show the search statistics. Since this line is prefixed with a symbol "#" it indicates that the proof was found. One such line is printed every iteration. The details are given as to how many inferences were performed, how many unit clauses were retained and resolved etc. The last entry on line 14 also indicates the average search speed.

The main line of the proof is printed next. The clauses are prefixed by indicators describing how and from which other clauses the current clause was derived. For instance, prefix "B" indicates a base clause, whereas "L" indicates a clause on the search line. Note that some inferences have two parents and others have one. Binary resolvent needs two parents whose numbers are thus given separated by a comma (e.g. line 24). Paramodulants also need two parents which are given separated by an arrow (e.g. line 23). This indicates "into" clause and "from" clause. Other clauses were produced by unary inference rules. The second letter in front of the clause indicates which unary inference rule was used (e.g. line 25 shows a factor). When indicating the parent, its clause number is given as well as the letter to specify which literal was involved.

Some of the literals are taken into brackets. This indicates a literal resolved by a unit clause retained in a hash table or a demodulator which was used to simplify the current clause. A bracketed letter in front of the clause enumerates all such instances (e.g. line 28). For every such instance a separate proof line will follow. Note that researched clause, ending auxiliary proof line, will always be prefixed with an "R" (e.g. line 31) whereas the places where such were used differentiate demodulators denoted by "D" (e.g. line 21) and hash table resolutions denoted by "H" (e.g. line 27).

Some special symbols also follow clauses. For instance "NC" indicates that this clause has at least one ancestor from the negated conclusion.

Note that the terms themselves are printed without parentheses. This is done to compact the output. We can nevertheless tell variables from constants since the former use letters starting from 'x'. We can also distinguish the subterms by knowing arities of all functions.

After the proof, a list of resolution rules which were used in the proof is given (lines 66-68) followed by a list of base clauses where the clauses which were actually used in the proof are pointed.

## A.2 Proof of Q14D10C.THM

```
GLIDE r7 Aug 1999 for (Q14D10C.THM)

P( equal D B C )
F( Ext ip p p1 p2 euc1 euc2 cont R a b )
C( p p1 p2 a b )
O( N(Q14D10C.THM) T900 D6/9 SB SEH SGH LN(30) LM LF LX EA PI PF DB GA
   GI HU HE SFS SFU SCS L7 V16 T9 )
S( LO PO BO FO RO 02 C(0,2) )
S( LO PO BO FO RO 00 C(0,0) )
E( 50 40 )
G( 40 27 )
# L( D1 T3 R(11743,2421,48,597) U(890,2296,10394) C(8047-0,12851)
     S(2205,0,0,0) G(15057,1531) V10465 )
```

```
Proof:
------
 02 ~equalExtababExtabba
 03 ~equalxy ~equalyz equalxz BF
S25(2a,3a) ~equalExtababx ~equalxExtabba BF
 [D0] BDM25(25a) ~equalRabx ~equalxExtabba [~equalRxyExtxyxy] BF
[H1] LA27(25a) # <~equalRabExtabba> BF

B17 equalRxyExtxyxy
R0 equalRxyExtxyxy

B0 BxyExtxyzu
B1 DxExtyxzuzu
B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dviwj BF
B19 DxRyxyx
B26 ~equalab NC
L27(26a,2g) ~Dabxy ~Dbzyu ~Davxw ~Dbvyw ~Babz ~Bxyu Dzvuw NC BF
 LD28(27c) ~Dabxy ~Dbzyu ~Dbxya ~Babz ~Bxyu Dzxua NC BF
  L29(28d,0a) ~Dabxy ~DbExtabzuyv ~Dbxya ~Bxyv DExtabzuxva NC BF
   L30(29b,1a) ~Dabxy ~Dbxya ~Bxyz DExtabyzxza NC BF
    LD31(30d) ~Dabxy ~Dbxya ~Bxya equalExtabyax NC BF
     [H2,H3] R1(31b,19a) equalExtabbaRab <~DabRabb> <~BRabba> NC BF

# L( D1 T4 R(19197,4029,60,1041) U(1778,3628,17785) C(13980-0,21424)
     S(3626,0,0,0) G(26927,2100) V13338 )
B18 BxyRxy
R3 BxyRxy

B19 DxRyxyx
R2 DxRyxyx

Resolution rules used:
----------------------
 BINRES ASSERT DISTANCE DEMOD UNIF_D UNIF_B

Used clauses:
-------------
>B0 BxyExtxyzu
>B1 DxExtyxzuzu
>B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dviwj BF
 B3 ~Bxyz ~Buvz Byipxyzvuu BF
 B4 ~Bxyz ~Buvz Bvipxyzvux BF
 B5 ~Bpp1p2
 B6 ~Bp1p2p
 B7 ~Bp2pp1
 B8 ~Dxyzz ~Duyuz ~Dvyvz equalyz Bxuv Buvx Bvxu BF
 B9 ~Bxyz ~Buyv equalxy Bxueuc1xuyvz BF
 B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
 B11 ~Bxyz ~Buyv equalxy Beuc1xuyvzzeuc2xuyvz BF
 B12 ~Dxyxz ~Dxuxv ~Bxyu ~Byvu Bzcontxyzvuvv BF
 B13(0a,1a) ~Dxyxz ~Dxuxv ~Bxyu ~Byvu ~equalcontxyzvuvi Dxvxi BF
 B14 ~Dxyzu ~Dzuvw Dxyvw BF
 B15 equalxExtyxzz
 B16 ~equalxExtyzuv Byzx
>B17 equalRxyExtxyxy
>B18 BxyRxy
>B19 DxRyxyx
 B20 ~equalxy equalyRxy
 B21 equalxRxx
 B22 ~equalxRyx equalyx
 B23 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
 B24 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
>B25(2a,3a) ~equalExtababx ~equalxExtabba BF
>B26 ~equalab NC
```

## A.3 Proof of Q21B2.THM

```
GLIDE r7 Aug 1999 for (Q21B2.THM)

P( equal D B C )
F( Ext ip p p1 p2 euc1 euc2 cont R a b c )
C( p p1 p2 a b c )
O( N(Q21B2.THM) T900 D6/9 SB SEH SGH LN(30) LM LF LX EA PI PF DB GA
   GI HU HE SFS SFU SCS L7 V16 T9 )
S( L0 P0 B0 F0 R0 O1 C(0,1) )
S( L0 P0 B0 F0 R0 O0 C(0,0) )
E( 60 50 )
G( 50 34 )
L( D1 T10 R(84171,10143,474,5130) U(11278,17586,66594) C(54401-0,85854)
   S(8841,7,0,0) G(88686,15632) V20423 )
* L( D2 T200 R(1752935,276436,4373,151332) U(19390,381689,1610721)
     C(1010599-0,1562649) S(247667,44,0,0) G(1744298,255958) V20926 )

Proof:
```

```
------
B3 ~Bxyz ~Buvz Byipxyzvuu BF
B30 ~Bxyz ~equalxz Buyz
B31 Babc
B32 Bbac
B33 ~equalab NC
LB34(30c) ~Bxyz ~equalxz equalzy
 L35(3c<-34c) ~Bxyz ~Buvz Bywu ~Biwipxyzvu ~equaliipxyzvu BF
  LB36(35d) ~Bxyz ~Buvz Bywu ~equalwipxyzvu BF
   L37(36a,31a) ~Bxyc Bbzx ~equalzipabcyx
    L38(37a,32a) Bbxb ~equalxipabcab
     LB39(38a) equalbx ~equalxipabcab
      [HO] L40(39a,33a) # <~equalaipabcab> NC

B4 ~Bxyz ~Buvz Bvipxyzvux BF
B31 Babc
B32 Bbac
L34(32a,4b) ~Bxyc Baipxycabx
 L35(34a,31a) Baipabcaba
  RB0(35a) equalaipabcab

# L( D1 T203 R(1781775,279679,4413,152764) U(20980,385253,1631704)
     C(1025060-0,1585755) S(251564,44,0,0) G(1766819,258549) V20906 )

Resolution rules used:
----------------------
 BINRES PARAMOD BETWEEN

Used clauses:
------------
 B0 BxyExtxyzu
 B1 DxExtyxzuzu
 B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dvivj BF
>B3 ~Bxyz ~Buvz Byipxyzvuu BF
>B4 ~Bxyz ~Buvz Bvipxyzvux BF
 B5 ~Bpp1p2
 B6 ~Bp1p2p
 B7 ~Bp2pp1
 B8 ~Dxyxz ~Duyuz ~Dvyvz equalyz Bxuv Buvx Bvxu BF
 B9 ~Bxyz ~Buyv equalxy Bxueuc1xuyvz BF
 B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
 B11 ~Bxyz ~Buyv equalxy Beuc1xuyvzzeuc2xuyvz BF
 B12 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Bzcontxyzwuvv BF
 B13(Oa,1a) ~Dxyzz ~Dxuxv ~Bxyu ~Bywu ~equalcontxyzwuvi Dxuxi BF
 B14 ~Dxyzu ~Dzuvw Dxyvw BF
 B15 equalxExtyxzz
 B16 ~equalxExtyzuv Byzx
 B17 equalRxyExtxyxy
 B18 BxyRxy
 B19 DxRyxyx
 B20 ~equalxy equalyRxy
 B21 equalxRxx
 B22 ~equalxRyx equalyx
 B23 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
 B24 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
 B25 ~Bxyz equalxy equalzExtxyyz
 B26 ~Dxyzu equalExtvwxyExtvwzu equalvw
 B27 equalExtxyxyExtxyyx equalxy
 B28 DxyxRRyxx
 B29 equalxRRxyy
>B30 ~Bxyz ~equalxz Buyz
>B31 Babc
>B32 Bbac
>B33 ~equalab NC
```

## A.4 Proof of Q31E3.THM

```
GLIDE r7 Aug 1999 for (Q31E3.THM)

P( equal D B C )
F( Ext ip p p1 p2 euc1 euc2 cont R b a d c )
C( p p1 p2 b a d c )
O( N(Q31E3.THM) T900 D6/9 SB SEH SGH LN(30) LM LF LX EA PI PF DB GA
   GI HU HE SFS SFU SCS L7 V16 T22 )
S( LO PO B2 FO R1 OO C(-2,1) )
S( LO PO BO FO RO OO C(0,0) )
E( 69 59 )
G( 59 43 )
# L( D1 T1 R(37,0,243,0) U(1153,4,38) C(25-0,3) S(15,0,0,0) G(16,6) V302 )

Proof:
------
B1 DxExtyxzuzu
B14 ~Dxyzu ~Dzuvw Dxyvw BF
 OO BxyExtxyzu
 01 equalbExtabpp1 ~DbExtabpp1dExtcdpp1 ~BabExtabpp1 NC
S42(0a,1a) equalbExtabpp1 ~DbExtabpp1dExtcdpp1 NC
L43(1a<-42a) Dbbpp1 ~DbExtabpp1dExtcdpp1 NC
 L44(43b,14c) Dbbpp1 ~DbExtabpp1xy ~DxydExtcdpp1 NC
  L45(44b,1a) Dbbpp1 ~Dpp1dExtcdpp1 NC
   L46(45b,1a) Dbbpp1 NC
    [HO] LD47(46a) # <equalpp1> NC

B39 ~equalpp1
R0 ~equalpp1

Resolution rules used:
----------------------
 BINRES PARAMOD DISTANCE

Used clauses:
------------
 B0 BxyExtxyzu
>B1 DxExtyxzuzu
 B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dviwj BF
 B3 ~Bxyz ~Buvz Byipxyzvuu BF
 B4 ~Bxyz ~Buvz Bvipxyzvux BF
 B5 ~Bpp1p2
 B6 ~Bp1p2p
 B7 ~Bp2pp1
 B8 ~Dxyxz ~Duyuz ~Dvyvz equalyz Bxuv Buvx Bvxu BF
 B9 ~Bxyz ~Buyv equalxy Bxueuc1xuyvz BF
 B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
 B11 ~Bxyz ~Buyv equalxy Beuc1xuyvzzeuc2xuyvz BF
 B12 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Bzcontxyzwuvv BF
 B13 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Dxwxcontxyzwuv BF
>B14 ~Dxyzu ~Dzuvw Dxyvw BF
 B15 equalxExtyxzz
 B16 ~equalxExtyzuv Byzx
 B17 equalRxyExtxyxy
 B18 BxyRxy
 B19 DxRyxyx
 B20 ~equalxy equalyRxy
 B21 equalxRxx
 B22 ~equalxRyx equalyx
 B23 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
 B24 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
 B25 ~Bxyz equalxy equalzExtxyyz
 B26 ~Dxyzu equalExtvwxyExtvwzu equalvw
 B27 equalExtxyxyExtxyyx equalxy
 B28 DxyxRRyxx
 B29 equalxRRxyy
 B30 ~Bxyz ~equalxz Buyz
 B31 ~Bxyz ~Byxz equalxy BF
 B32 ~Bxyz ~Bxzy equalyz BF
 B33 ~Bxyz ~Byuz Bxyu BF
 B34 ~Bxyz ~Bxzu Byzu BF
 B35 ~Bxyz ~Byzu Bxzu equalyz BF
 B36 ~Bxyz ~Byzu Bxyu equalyz BF
 B37 ~Bxyz ~Byuz Bxuz BF
 B38 ~Bxyz ~Bxzu Bxyu BF
>B39 ~equalpp1
 B40 ~equalp1p2
 B41 ~equalpp2
>B42(0a,1a) equalbExtabpp1 ~DbExtabpp1dExtcdpp1 NC
```

## A.5 Proof of Q38I2B.THM

```
GLIDE r7 Aug 1999 for (Q38I2B.THM)

P( equal D B C )
F( Ext ip p p1 p2 eucl euc2 cont ins R a b c d e )
C( p p1 p2 a b c d e )
O( N(Q38I2B.THM) T900 D6/9 SB SEL SGH LN(30) LM LF LX UEA DB UGA GI
   HU HE SFS SFU SCS L7 V16 T15 )
S( L0 P0 B3 F0 R0 O0 C(-3,0) )
S( L0 P0 B0 F0 R0 O0 C(0,0) )
E( 85 83 )
G( 83 67 )
L( D1 T115 R(811178,43027,0,0) U(3503319,27552,1050910)
   C(381144-1789990,390011) S(4901,8,0,0) G(0,96870) V8270 )
O( N(Q38I2B.THM) T900 D6/9 SB SEL SGH LN LM LF LX UEA DB UGA GI
   HU HE SFS SFU SCS L7 V16 T15 )
L( D2 T279 R(1950592,137092,0,0) U(4001458,67866,1346962)
   C(1359536-3252592,430628) S(27671,8,0,0) G(0,160973) V8059 )
# L( D3 T562 R(3352210,277537,0,0) U(5159800,145335,1726450)
     C(2741408-5790385,454735) S(56611,8,0,0) G(0,331115) V7047 )

Proof:
------
```

```
B0 BxyExtxyzu
B28 ~equalxExtyzuv Byzx
B36 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
B48 ~Bxyz ~Byzu Bxyu equalyz BF
B54 ~equalxExtyxpp1
B62 equalinsxyzuExtExtyxpp1xzu
B66 ~Bdinsdeabe NC
L67(66a,28b) ~equaleExtdinsdeabxy NC
 L68(67a,36e) ~Bxye ~BxyExtdinsdeabzu ~DyeyExtdinsdeabzu equalxy NC
  L69(68b,48c) ~Bxye ~DyeyExtdinsdeabzu equalxy ~Bxyv ~ByvExtdinsdeabzu
               equalyv NC BF
   L70(69d,28b) ~Bxye ~DyeyExtdinsdeabzu equalxy ~ByvExtdinsdeabzu equalyv
                ~equalvExtxywi NC
    L71(70a,0a) ~DxexExtdinsdeabyz equalExtexuvx ~BxwExtdinsdeabyz equalxw
                ~equalwExtExtexuvxij NC
     [H0] L72(71c,0a) ~DdedExtdinsdeabxy equalExtedzud
                      ~equalinsdeabExtExtedzudvw <equaldinsdeab> NC
      L73(72b,54a) ~DdedExtdinsdeabxy ~equalinsdeabExtExtedpp1dzu NC
       [H1] L74(73b,62a) # <~DdedExtdinsdeabxy> NC

B0 BxyExtxyzu
B14 ~equalxy ~equalyz equalxz BF
B15 ~equalxy ~Bxzu Byzu
B27 equalxExtyxzz
B66 ~Bdinsdeabe NC
L67(66a,15c) ~equalxd ~Bxinsdeabe NC
 L68(67a,14c) ~Bxinsdeabe ~equaldy ~equalyx NC BF
  L69(68a,0a) ~equaldx ~equalxExteinsdeabyz NC
   R0(69b,27a) ~equaldinsdeab NC

B0 BxyExtxyzu
B1 DxExtyxzuzu
B26 ~Dxyzu ~Dzuvw Dxyvw BF
B35 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
B63 Dxyzinszuxy
B64 Babc
B65 Dacde
L67(35e,26a) ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw ~Dzwij Dxvij BF
 L68(67c,0a) ~Dxyzu ~DyExtxyvwui ~Bzui ~Dzijk DxExtxyvwjk BF
  L69(68b,1a) ~Dxyzu ~Bzuv ~Dzvwi DxExtxyuvwi BF
   L70(69b,64a) ~Dxyab ~Daczu DxExtxybczu BF
    L71(70a,63a) ~Dacxy DzExtzinszuabbcxy
     R1(71a,65a) DxExtxinsxyabbcde
```

```
# L( D1 T619 R(4007534,311774,0,0) U(6305263,166028,1920955)
     C(3023496-5792975,804921) S(60870,16,0,0) G(0,406703) V7634 )

Resolution rules used:
----------------------
 BINRES UNIF_D

Used clauses:
------------
>B0 BxyExtxyzu
>B1 DxExtyxzuzu
 B2 ~Dxyzu ~Dyvuw ~Dxizj ~Dyiuj ~Bxyv ~Bzuw equalxy Dviwj BF
 B3 ~Bxyz ~Buvz Byipxyzvuu BF
 B4 ~Bxyz ~Buvz Bvipxyzvux BF
 B5 ~Bpp1p2
 B6 ~Bp1p2p
 B7 ~Bp2pp1
 B8 ~Dxyxz ~Duyuz ~Dvyvz equalyz Bxuv Buvx Bvxu BF
 B9 ~Bxyz ~Buyv equalxy Bxueuc1xuyvz BF
 B10 ~Bxyz ~Buyv equalxy Bxveuc2xuyvz BF
 B11 ~Bxyz ~Buyv equalxy Beuc1xuyvzzeuc2xuyvz BF
 B12 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Bzcontxyzwuvv BF
 B13 ~Dxyxz ~Dxuxv ~Bxyu ~Bywu Dxwxcontxyzwuv BF
>B14 ~equalxy ~equalyz equalxz BF
>B15 ~equalxy ~Bxzu Byzu
 B16 ~equalxy ~Bzxu Bzyu
 B17 ~equalxy ~Bzux Bzuy
 B18 ~equalxy ~Dxzuv Dyzuv
 B19 ~equalxy ~Dzxuv Dzyuv
 B20 ~equalxy ~Dzuxv Dzuyv
 B21 ~equalxy ~Dzuvx Dzuvy
 B22 ~equalxy equalinsxzuvinsyzuv
 B23 ~equalxy equalinszxuvinszyuv
 B24 ~equalxy equalinszuxvinszuyv
 B25 ~equalxy equalinszuvxinszuvy
>B26 ~Dxyzu ~Dzuvw Dxyvw BF
>B27 equalxExtyxzz
>B28 ~equalxExtyzuv Byzx
 B29 equalRxyExtxyxy
 B30 BxyRxy
 B31 DxRyxyx
 B32 ~equalxy equalyRxy
 B33 equalxRxx
 B34 ~equalxRyx equalyx
>B35 ~Dxyzu ~Dyvuw ~Bxyv ~Bzuw Dxvzw BF
>B36 ~Bxyz ~Bxyu ~Dyzyu equalxy equalzu BF
 B37 ~Bxyz equalxy equalzExtxyyz
 B38 ~Dxyzu equalExtvwxyExtvwzu equalvw
 B39 equalExtxyxyExtxyyx equalxy
 B40 DxyxRRyxx
 B41 equalxRRxyy
 B42 ~Bxyz ~equalxz Buyz
 B43 ~Bxyz ~Byzz equalxy BF
 B44 ~Bxyz ~Bxzy equalyz BF
 B45 ~Bxyz ~Byuz Bxyu BF
 B46 ~Bxyz ~Bxzu Byzu BF
 B47 ~Bxyz ~Byzu Bxzu equalyz BF
>B48 ~Bxyz ~Byzu Bxyu equalyz BF
 B49 ~Bxyz ~Byuz Bxuz BF
 B50 ~Bxyz ~Bxzu Bxyu BF
 B51 ~equalpp1
 B52 ~equalp1p2
 B53 ~equalpp2
>B54 ~equalxExtyxpp1
 B55 DxExtyxpp1zExtuzpp1
 B56 ~Bxyz ~Buvz ~Bxwu Bwipvipxwuvzxyzz Byipvipxwuvzxyzv BF
 B57 ~Bxyz ~Dxzxu ~Dyzyu equalxy equalzu BF
 B58 ~Dxyzu ~Dxvzw ~Dxizj ~Dviwj ~Bxyv ~Bzuw Dyiuj BF
 B59 ~Bxyz ~Buvw ~Dxyuv ~Dxzuw Dyzvw BF
 B60 ~Dxyzu ~Dyvuw ~Dxizj ~Dviwj ~Bxyv ~Bzuw Dyiuj BF
 B61 ~Bxyz ~Dxyxu ~Dzyzu equalyu BF
>B62 equalinsxyzuExtExtyxpp1xzu
>B63 Dxyzinszuxy
>B64 Babc
>B65 Dacde
>B66 ~Bdinsdeabe NC
```

# Index