CLARKSON UNIVERSITY

Automated Theorem Proving Using SAT

A Ph.D. dissertation
by

Ralph Eric McGregor

Department of Computer Science

Submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

(Computer Science)

2011

Accepted by the Graduate School

_____
Date

_____
Dean

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

The undersigned have examined the proposal entitled

Automated Theorem Proving Using SAT

presented by

Ralph Eric McGregor

a candidate for the degree of

Doctor of Philosophy (Computer Science),

and here by certify that it is worthy of acceptance.

Examining Committee:

Date	Christopher Lynch (Advisor)
Date	Daqing Hou
Date	James Lynch
Date	Alexis Maciel
Date	Christino Tamon

Abstract

The first refutationally complete inference systems for first-order logic, called instance-based systems, were based on Herbrand's theorem which implies that first-order logic satisfiability can be reduced to propositional logic satisfiability (SAT). Out of this line of research came the landmark SAT solving DPLL algorithm. Soon after DPLL made its debut, Robinson introduced the simple combinatorial resolution rule which detracted interest in instance-based systems. Recently, with the increase in computational power of the personal computer, there has been renewed interest in systems for first-order logic theorem proving that utilize SAT solvers. Here, we present three novel solutions for the first-order logic validity problem that utilize SAT.

As our first solution, we reduce first-order logic validity to SAT, by encoding a proof of first-order logic unsatisfiability in propositional logic and use a SAT solver to determine if the encoding is satisfiable. Specifically we encode a closed connection tableaux proof of unsatisfiability. A satisfiable propositional encoding implies validity of the first-order problem. We provide an encoding using SAT, provide soundness and completeness proofs, and discuss our implementation, called CHEWTPTP-SAT, along with results. We also give an encoding in SMT, provide soundness and completeness proofs, discuss our implementation, CHEWTPTP-SMT, and discuss when then encoding in SMT may be better than an encoding in SAT.

The second solution is an inference system, called SIG-Res, which combines SInst-Gen (which utilizes SAT) with resolution into a single sound and refutationally com-

plete system. We allow a set of clauses, S, to be distributed among two sets P and R in any combination so long as $S = P \cup R$. SInst-Gen is run on P with conclusions added to P, resolution is run on R with conclusions added to R, and resolution is run on $P \times R$ with conclusions added to P. Factoring is also used on all conclusions to resolution inferences. Since any distribution is allowed, this system permits a spectrum of choices. At the ends of the spectrum, for instance if $P = S$, the system is simply a SInst-Gen solver and if $R = S$, the solver is a resolution solver. We give the inference rules for SIG-Res, provide soundness and completeness proofs, and discuss an implementation, called Spectrum, along with experimental results. We also discuss a newer implementation, called EVC3, which combines the SMT solver named CVC3 with the equational theorem prover, E.

Our third solution establishes a framework, called $\Gamma + \Lambda$, which allows a wide range of first-order logic calculi to be combined into a single sound and refutationally complete system. This framework can be used to combine instance generation methods that use SAT with other inference systems. In order to combine two systems, Γ and Λ, into a single system, we require Γ to be productive and require Λ to have both the lifting and total-saturation properties. This framework allows a set of clauses to be distributed among two sets, P and R, like SIG-Res, so that Γ can be used on P and Λ can be used on R. A limited amount of information is passed between the systems to establish completeness of the combined system. We give the inference rules for $\Gamma + \Lambda$, establish soundness and completeness, and show how Inst-Gen-Eq and superposition can be combined in this framework.

Acknowledgments

During these last six years, while working toward the degree of Doctor of Philosophy, I have been blessed to have the support of many truly great family members, colleagues and friends. I thank you all! You have taught me perseverance, modesty and compassion. You have provided encouragement, knowledge, respect and love. You have given me opportunities that few in this world have been given. For all this, I am indebted to you.

Ahmad Almomani
Susan Blanchard
Elmer Deshane
Howard Deshane
Deena & Brian Donnelly
Michael Felland
Claudette Foisy
Dr. Kathleen Fowler
Michael Fowler
Dr.s Illona & Donald Ferguson
Dr. Daqing Hou
Dr. Christopher Lynch
Dr. James Lynch
Dr. Alexis Maciel
Dr. Jeanna Matthews
Judge Earl McBride
Pamula & Ralph McGregor
Gary McGregor
Sara Morrison
The Brothers of Phi Kappa Sigma
Dr. Joseph Skufca
Janice Searleman
Cindy Smith
Dr. Christino Tamon
Elizabeth Thomas
Dean Peter Turner

Thank you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reasoning is the act of learning from the use of models and information [83]. We reason consciously and subconsciously. We use reason to form intuition, to form habit and we reason to determine causality, a necessity for survival. We also reason to understand relationships among entities we perceive and conceive to better understand the universe around us.

Since antiquity we have also sought to understand the nature of valid reasoning itself in the study of logic. Since Aristotle [1], attempts have been made to model reasoning in an effort to mechanize reasoning with the goal of eliminating the errors found in human reasoning. More recently, with the advent of the computer many have sought to automate certain forms of reasoning.

Many different logics have been developed over time including propositional, temporal, modal, Hoare, first-order logic and higher-order logics. In 1930, G*ö*del [19] proved that there exist complete and sound inference systems for first-order logic and in 1936 and 1937, Church and Turing [22, 24], respectively, proved independently that first-order validity is undecidable, thereby establishing that first-order validity is semi-decidable. So the best that we can do is construct a sound and complete inference system for first-order logic, that when given a valid formula, will eventually halt and output an indication of validity.

However if an invalid formula is given, the system may never halt.

One subset of the automated reasoning (AR) community focus' on automated theorem proving (ATP), that is, the study of systems that semi-decide the first-order validity problem.

Since ϕ is valid iff $\neg\phi$ is unsatisfiable, many ATP systems alternatively seek to prove unsatisfiability rather than validity. A formula is unsatisfiable if no model exists which makes the formula true. If a system, when given an unsatisfiable problem, will eventually halt and output unsatisfiable, we call the system *refutationally complete*.

Today's refutationally complete ATP systems are based on a wide array of techniques. Some seek to provide a proof of unsatisfiability (e.g. resolution and tableaux proof encoding) while others attempt to prove unsatisfiability by showing no model exists (e.g. instance generation). These ATP systems can be used as general theorem provers or as back end provers in larger software systems such as software verification tools.

Below we discuss some common techniques for first-order theorem proving, then discuss our contributions in ATP and our current work and conclude with an outline of this dissertation.

1.1 Techniques for Automated Theorem Proving

Most modern automated theorem proving systems are based on superposition, semantic selection instance generation, or tableaux methods. In this section we describe each of these methods and provide a brief history of the development of each.

1.1.1 Instance Generation Based Systems

Instance generation techniques seek to find a set of ground (variable free) instances for the input problem which are unsatisfiable when viewed as a propositional logic formula via a SAT solver. If the SAT solver returns unsatisfiable, the input problem is unsatisfiable.

If the SAT solver returns satisfiable and new instances can be generated (according to the calculus) they are added and the process is repeated, otherwise the input problem is found to be satisfiable.

One of the first implemented general first-order logic theorem provers was created in 1960 by Davis and Putnam [29]. Their procedure was based on Herbrand's Theorem [21] which shows that first-order logic unsatisfiability can be reduced to propositional logic unsatisfiability.

We say the Herbrand Universe of a formula ϕ is the set of all ground terms that can be constructed using the function symbols and constants in ϕ (if no constant is defined we include a distinguished constant). For example, if P is a predicate symbol, f is a function symbol, a is a constant and x is a variable, the Herbrand Universe for $P(f(x), a)$ is $\{a, f(a), f(f(a)), f(f(f(a))), ...\}$. Herbrand's Theorem can be stated as follows: a first-order formula ϕ is unsatisfiable if and only if there exists a finite set of ground instances (replacing variables with elements of the Herbrand Universe) of ϕ is unsatisfiable. Since all of the instances in the set being checked for satisfiability are ground, the formula in this set can be viewed as a propositional formula whose satisfiability can be checked by a propositional logic decision procedure.

Suppose for example

$$\phi = ((p(x) \implies q(x)) \wedge p(a)) \implies q(a) \tag{1.1}$$

To prove ϕ is valid, we first convert $\neg\phi$ to conjunctive normal form as shown in Formula 1.2.

$$\neg\phi = (\neg p(x) \vee q(x)) \wedge p(a) \wedge \neg q(a) \tag{1.2}$$

Then we seek to show that $\neg\phi$ is unsatisfiable. In this trivial problem, to determine the unsatisfiability of $\neg\phi$ using Herbrand's Theorem we simply need to create a single instance of $\neg\phi$, namely, $\neg\phi'$ [Formula 1.3].

$$\neg\phi' = \quad (\neg p(a) \vee q(a)) \wedge p(a) \wedge \neg q(a) \tag{1.3}$$

$$(\neg P \vee Q) \wedge P \wedge \neg Q \tag{1.4}$$

We derive $\neg\phi'$ from $\neg\phi$ by replacing all instances of the variable x by the constant a. Since $\neg\phi'$ is unsatisfiable when viewed as a propositional formula [Formula 1.4], by Herbrand's Theorem $\neg\phi$ is unsatisfiable as a first-order formula, thus ϕ is valid.

Davis and Putnam's procedure (although naively) incrementally adds all possible ground instances to the set being checked for propositional satisfiability until unsatisfiability is detected. Although their choice to consider all ground instances was found to be unnecessary, their choice of using conjunctive normal form (CNF) along with their procedure for satisfiability testing at the propositional level, and the improved procedure by Davis, Logeman and Loveland [34], now called the DPLL procedure, was revolutionary and is the basis for the most efficient satisfiability (SAT) solvers today. Today's SAT solvers based on the DPLL procedure can efficiently handle tens of thousands of variables and clauses [108].

Since Davis and Putnam's work, there have been a great many advances in theorem provers based on Herbrand's Theorem, which are called *instance-based* theorem provers. One line of research has been in *saturation-based instance generation* methods [106]. These methods, like Davis and Putnam's method, repeatedly call upon a SAT solver to determine the satisfiability of the current set of ground instances. If unsatisfiable the solver halts and indicates such, otherwise the set may be augmented with additional instances and again checked for satisfiability.

Notable in this line of research is the work by Prawitz [30] who showed that not all elements in the Herbrand Universe need be used. Elements in the Herbrand Universe were chosen using a matching scheme among complementary literals [53]. This technique evolved into what is now known as unification. In [35], Davis combined the matching technique of Prawitz with the techniques in his previous paper.

A lull transpired in this line of research, while the focus was on resolution based systems, until the late eighties when computational power began to increase significantly and methods based on DPLL and SAT again attracted appeal. In 1988 Jeroslow [63] published a new saturation-based instance generation method called *Partial Instantiation* (PI). In this work he proved that not all variables in the non-ground instances need be replaced by terms in the Herbrand Universe. Substitutions can be identified using the matching technique and those variables that are not involved in the unification can simply be mapped to a distinguished constant, the Jeroslow constant. Hooker, Rago, Chandru and Shrivastava are noteworthy for developing the first complete PI method for the full first-order predicate calculus called *Primal PI* [99] which is now known as *instance generation with semantic selection*. Ganzinger and Korovin, among many other contributions, formalized and proved the completeness of the instance generation inference rule (Inst-Gen) and instance generation with semantic selection and hyper inferences (SInst-Gen) in [106].

Saturation-based instance generation methods perform especially well on problems that are *near* to propositional logic where the efficiency of the SAT solver is exploited. One such class of first-order logic problems is Effectively Propositional Logic (EPR), also know as the Bernays-Schönfinkel class [18]. Formulas in this class have the form $\exists\forall\phi$ where ϕ is quantifier-free and function-free. As seen in the results from the annual Conference on Automated Deduction (CADE) ATP System Competition (CASC) [131], the saturation-based instance generation theorem prover iProver [142] has won the EPR category since 2008. It should be noted that the success of iProver, and thus a defense of instance generation solvers as a viable general first-order theorem proving technique, has not only been in the

EPR class, but iProver has also ranked in the top 5 in the FOF (first-order formula) and CNF (conjunctive normal form) divisions, the classes of general first-order formulas, in the 3 most recent CASC competitions. This is evidence that techniques in saturation based instance generation are viable and worthy of continued research.

1.1.2 Resolution Based Systems

Another well known line of research in general first-order logic theorem proving began with Robinson's landmark paper [36] which describes his resolution principle and unification algorithm which computes the most general unifier (mgu) between two atoms. Here Robinson developed a refutationally complete inference system using a single combinatorial rule (Figure 1.1) without the need for an axillary propositional logic solver like the instance generation methods that proceeded it required. With resolution (a generalization of Modus Ponens) the goal is to derive a proof of the empty clause.

$$\frac{L \vee \Gamma \qquad \neg K \vee \Delta}{(\Gamma \vee \Delta)\sigma} \text{ (Resolution)}$$

where $\sigma = \mathrm{mgu}(L, K)$

$$\frac{L \vee K \vee \Gamma}{(L \vee \Gamma)\sigma} \text{ (Factoring)}$$

where $\sigma = \mathrm{mgu}(L, K)$

Figure 1.1: Resolution and Factoring Inference Rules

A proof of Formula 1.2 using resolution can be given by

$$\dfrac{\dfrac{\neg p(x) \vee q(x) \qquad p(a)}{q(a)}\, x \mapsto a \qquad \neg q(a)}{\bot}$$

where $\bot$ denotes the empty clause (or falsum) and $x \mapsto a$ denotes the replacement of a for x in the premises.

A key refinement of resolution called *ordered resolution* (formerly A-ordered resolution) was made in [42, 92] which restricts the search space significantly. With ordered resolution, resolution inferences are only made when complementary maximal literals, based on an ordering of terms, from different clauses are unifiable. Ordered resolution can be efficient in practice, because it tends to produce literals in the conclusion of an inference that are smaller than in the premises. This is not always the case however, because the most general unifier may prevent that, but it often happens in practice. For the simplest example, consider a set of clauses consisting of one non-ground clause $C = \neg P(x) \vee P(f(x))$ and any number of ground clauses. Any ordered resolution inference among two ground clauses will produce another ground clause which does not introduce any new literals. Any ordered resolution inference between a ground clause and C will produce a new ground clause where an occurrence of the symbol f has disappeared. This will clearly halt. This reduction property does not hold for instance generation methods. If this set of clauses is fed to an instantiation-based prover, it may run forever, depending on the model created by the SAT solver. In our experiments, this does run forever in practice.

$$
\begin{array}{rcll}
 & \rightarrow & x \simeq x & \text{(reflexivity)} \\
x \simeq y & \rightarrow & y \simeq x & \text{(symmetry)} \\
x \simeq y \wedge y \simeq z & \rightarrow & x \simeq z & \text{(transitivity)} \\
x_1 \simeq y_1 \wedge ... \wedge x_n \simeq y_n & \rightarrow & f(x_1, ..., x_n) \simeq f(y_1, ..., y_n) & \text{(monotonicity I)} \\
x_1 \simeq y_1 \wedge ... \wedge x_n \simeq y_n & & & \\
\wedge P(x_1, ..., x_n) & \rightarrow & P(y_1, ..., y_n) & \text{(monotonicity II)}
\end{array}
$$

Figure 1.2: Equality Congruence Axioms

Many other refinements have been made to Robinson's resolution inference system, e.g. semantic resolution [38] and set of support resolution [48]. As early as 1960 [32], researchers have also constructed ways to handle equality directly in the calculus. A naive approach to handling equality would be to include the congruence axioms [1] (Figure 1.2) for the equality predicate $\simeq$ in the problem and use resolution and factoring inferences.

[1] One monotonicity I axiom is added for each non-constant n-ary function symbol in the language and one monotonicity II axiom is added for each predicate symbol in the language

This however causes an explosion in the search space.

To avoid this, in 1969, Wos and Robinson in [41] introduced the inference rule called paramodulation (Figure 1.3) that has been a cornerstone of modern resolution based theorem provers that handle equality. In their work they proved that the paramodulation inference rule along with resolution, factoring and additional function reflexivity axioms [2]form a refutationally complete system.

$$\frac{L[u]_p \vee \Gamma \qquad l \simeq r \vee \Delta}{(L[r]_p \vee \Gamma \vee \Delta)\sigma}\ \text{Paramodulation}\ ^{3}$$

where $\sigma = \mathrm{mgu}(l, u)$

Figure 1.3: Paramodulation Inference Rule

Take for example the problem

$$(f(x) = x) \wedge P(f(a)) \wedge \neg P(a) \tag{1.5}$$

A paramodulation proof for Formula 1.5 can be written as

$$x \mapsto a\ \frac{\dfrac{f(x) = x \qquad P(f(a))}{P(a)} \qquad \neg P(a)}{\bot}$$

Brand showed in [47] (later by Peterson in [54]) that paramodulation, resolution and factoring alone form a refutationally complete system and that paramodulation *into variables*, which is allowable in Wos and Robinson's work, is not necessary [47].

In 1970, working independently from Wos and Robinson, Knuth and Bendix published the first ordered paramodulation calculus for systems of equations (the word problem) called Knuth-Bendix completion [44]. Later, in 1987, Rusinowitch and Hsiang [60] (and others later) extended Knuth-Bendix completion to unfailing completion.

[2]A functional reflexivity axiom $f(x_1, ..., x_n) \simeq f(x_1, ..., x_n)$ is required for each n-ary function symbol in the language

[3]$L[u]_p$ denotes the literal L containing the subterm u at position p

Peterson, in 1983, combined the works of Wos and Robinson with that of Knuth and Bendix to produce what is known as *ordered paramodulation* (Figure 1.4), a refutationally complete system for first-order logic that uses term orderings to restrict paramodulation inferences [54].

$$\frac{L[u]_p \vee \Gamma \qquad l \simeq r \vee \Delta}{(L[r]_p \vee \Gamma \vee \Delta)\sigma} \text{ (Ordered Paramodulation)}$$

where

1. $\sigma = mgu(u, l)$
2. u is not a variable
3. $r\sigma \not\succeq l\sigma$

Figure 1.4: Ordered Paramodulation Inference Rule

In 1991, Pais and Peterson [67] introduced *maximal paramodulation* (Figure 1.5) which utilizes term orderings as well as orderings on literals [91].

$$\frac{L[u]_p \vee \Gamma \qquad l \simeq r \vee \Delta}{(L[r]_p \vee \Gamma \vee \Delta)\sigma} \text{ (Maximal Paramodulation)}$$

where

1. $\sigma = mgu(u, l)$
2. u is not a variable
3. $r\sigma \not\succeq l\sigma$
4. $L[u]_p\sigma$ is maximal w.r.t. $\succ$ in $(L[u]_p \vee \Gamma)\sigma$
5. $(l \simeq r)\sigma$ is maximal w.r.t. $\succ$ in $(l \simeq r \vee \Delta)\sigma$

Figure 1.5: Maximal Paramodulation Inference Rule

The next refinement called *superposition* considers all literals as equations. Here the selected literal from the left premise is of the form $s \simeq t$ or $s \not\simeq t$. We include the case for the former in Figure 1.6. In 1988, Zhang and Kapur were the first to propose this rule [62]. Bachmair and Ganzinger showed in [64] that Zhang and Kapur's system was incomplete in the presence of tautology elimination, and provide a complete superposition calculus which includes the additional equality factoring inference rule.

$$\frac{s[u]_p \simeq t \vee \Gamma \qquad l \simeq r \vee \Delta}{(s[r]_p \simeq t \vee \Gamma \vee \Delta)\sigma} \text{ (Superposition)}$$

where

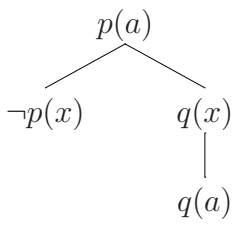
1. $\sigma = mgu(u, l)$
2. u is not a variable
3. $t\sigma \not\succeq s[u]\sigma$
4. $r\sigma \not\succeq l\sigma$
5. $(s[u]_p \simeq t)\sigma$ is maximal w.r.t. $\succ$ in $(s[u]_p \simeq t \vee \Gamma)\sigma$
6. $(l \simeq r)\sigma$ is maximal w.r.t. $\succ$ in $(l \simeq r \vee \Delta)\sigma$

Figure 1.6: Superposition Inference Rule

Most modern implementations of resolution-based theorem provers [102, 145] implement some form of superposition. These have been shown to be the top performers in the CASC competition [131]. Although resolution-based methods appear to be generally more efficient in practice, there are some classes of problems that are suited better for instantiation-based methods. As stated above, instantiation-based methods work especially well on problems that are close to propositional problems, because then the key technique is the DPLL procedure in the SAT solver. On the EPR class of first-order logic problems resolution methods may in fact run forever.

1.1.3 Tableaux Proofs

A tableau is a finitely branching tree with nodes labeled by formulas. Tableaux have been studied as far back as 1955 in works by Beth [25] and Hintikka [26] as a way to represent refutation proofs in first-order predicate calculus theorem proving. For example a proof for Formula 1.2 can be represented by the following tableau:



In standard refutational theorem proving we attempt to prove the unsatisfiability of a set of clauses and allow an unbounded number of renamed instances of each clause. In rigid theorem proving, only one instance of each clause is allowed. The tableau above is an example of a rigid proof (in the form of a rigid tableau). Rigid theorem proving has been studied as early as [45, 50] and can be used to solve the general theorem proving problem as described in [88]. To solve the general theorem proving problem we can repeatedly call a rigid theorem proving solver, each time adding additional renamed instances of the original clauses, until a proof is found [126].

1.2 Contributions in Automated Theorem Proving

Our work has focused on three novel methods for general first-order theorem proving that utilize SAT. The first line of research solves the general first-order logic theorem proving problem by encoding the existence of rigid tableaux proofs in propositional logic and using an external SAT solver to determine the satisfiability of the encodings. We extended this work by encoding the existence of rigid tableaux proof in propositional logic *modulo theories* with the hopes of reducing the size of the encoding. The second line of research combined the instance generation and resolution inferences systems into one refutationally complete inference system. The third line, provides a general framework for combining inference systems and show that $\mathrm{Inst\text{-}Gen\text{-}Eq}$ and superposition can be combined in this framework. Below we give a general overview of each of these systems and describe my personal contributions.

1.2.1 Encoding First-Order Proofs in SAT

Rather than directly computing a first-order logic refutation proof, one can *encode* the *existence* of a first-order logic refutation proof in propositional logic and establish the satisfiability of the encoding using a SAT solver. In this case, if the SAT solver returns satisfiable

then a refutation proof exists for the first-order formula. This is an attractive solution to the general first-order theorem proving problem, especially with the advancement of SAT solving technology and computing power, since the majority of computational work is done by the SAT solver.

In [121], Prestwich and Lynce encode a resolution proof, although in that paper propositional proofs were encoded and not first order proofs. Below and in [126], a joint work with DeShane, Hu, Jablonski, Lin and Lynch, we proposed encoding rigid connection tableaux proofs of first-order predicate calculus in propositional logic as an incremental approach to general first-order theorem proving. We also provide a proof of its completeness, describe our implementation called ChewTPTP-SAT and give results.

ChewTPTP-SAT is a sound and complete first-order theorem prover. While we are able to identify problems that ChewTPTP-SAT is able to solve and other theorem provers are unable to solve, overall, ChewTPTP-SAT is not currently competitive as a general first-order theorem prover. Below we identify reasons for this and suggest one modification that we believe will make our implementation much more competitive.

While developing this method I was involved in many aspects of our work. Specifically, I contributed to the development of the encoding, wrote the Lex/Yacc TPTP parser for the implementation, ran the experimental tests and compiled the results, wrote the soundness and completeness proofs and authored the paper that we submitted for publication.

1.2.2 Encoding First-Order Proofs in SMT

In [128], a joint work with Bongio, Katrak, Lin and Lynch, we state that our original ChewTPTP-SAT implementation [126] performed well on some problems, but some of the encodings created huge sets of clauses. Some parts of our encoding represented *choices* made, such as which clause to extend each literal with, and other parts of our encoding represented *deterministic procedures*, such as deciding the consistency of unification constraints and deciding the acyclicity of the DAG which verifies that a particular property

holds of the DAG. In our solver, ChewTPTP-SAT, we had an eager encoding of unification and acyclicity. In experimental results with Horn clauses (clauses containing no more than one negative literal), approximately 99% of the clauses generated were encoding the deterministic procedures, and only about 1% represented the choices.

We decided the implementation would be more efficient if unification and acyclicity were encoded lazily and implemented these changes in ChewTPTP-SMT. It makes sense to expresses choices involved in building the tableau using SAT, and verification of unification and acyclicity using underlying theories. Therefore, we chose to encode our problem as Satisfiability modulo Theories (SMT) [119] and replaced Minisat[108] with the SMT solver Yices [147].

Below we discuss our encoding for a tableaux proof in propositional logic modulo Theories (datatypes and arithmetic). We discuss our implementation called ChewTPTP-SMT, an extention of ChewTPTP-SAT, which is also a complete and sound first-order theorem prover. We show that for Horn clauses the encoding is smaller and the implementation is faster than our previous encoding, but for non-Horn problems the performance was worse than our previous encoding. We also describe a heuristic for when to use theories when encoding a problem. Although it can not be considered a competitive theorem prover in its current state, we believe that improvements can be made in both the encoding and in the implementation that will make it a viable solution to the first-order theorem proving problem and discuss these below.

Similar to the work previously discussed, I personally ran the experimental tests and compiled the results, wrote the soundness and completeness proofs and authored the paper that we submitted for publication. In addition, using CHEWTPTP-SAT as a code base, I wrote a significant amount of the new code for CHEWTPTP-SMT.

1.2.3 Combining Instance Generation and Resolution

Both instance generation and resolution methods have a long history in the automated reasoning community and are the basis for the most advanced modern ATP systems. Both types of systems have their strengths and weaknesses. As we saw above, instance-based provers may run forever on the satisfiable problem consisting of a single non-ground clause $\neg P(x) \vee P(f(x))$ combined with any number of ground clauses and resolution-based systems may run forever on problems in EPR.

Below and in [136], a joint work with Lynch, we showed that we can combine both instance generation and resolution into a single refutationally complete inference system with the aim of getting the best of both methods. We proposed the inference systems SIG-Res which combines semantic selection instance generation (SInst-Gen) with ordered resolution and discuss our implementation called Spectrum.

In SIG-Res, each clause in a given problem is determined to be, by some heuristic, an instance generation clause and added to the set P or a resolution clause and placed in the set R. Clauses from P are given to a SAT solver and inferences among them are treated as in SInst-Gen, while inferences that involve clauses in R are resolution inferences. Unsatisfiability is witnessed by the unsatisfiability of P under Inst-Gen or the unsatisfiability of R under resolution. Satisfiability is witnessed by a saturated system with P being satisfiable under Inst-Gen and R not containing the empty clause.

Spectrum is a sound and complete implementation of SIG-Res. More work however, needs to go into its development in order to make it competitive with state-of-the-art theorem provers. Specifically, we need to include more redundancy elimination techniques. Even though our implementation is not as sophisticated as state-of-the-art provers, we identify a class of problems that when run on Spectrum are solved faster by $SIGRes$ than by $SInstGen$ or resolution alone. We discuss these results and suggest improvements below.

Personal contributions to this work are the soundness and completeness proofs for SIG-Res, primary authorship of the published paper and sole developer of Spectrum.

1.2.4 $\Gamma + \Lambda$

In joint work, also with Lynch, we have developed a framework called $\Gamma + \Lambda$ which allows two different (or possibly the same) inference systems to be combined into a single system. The only conditions on Γ and Λ are that they both be sound and refutationally complete, Γ be productive and Λ have the lifting and total-saturation properties. When these conditions are met, the resulting system is sound and refutationally complete.

Like $\mathrm{SIG\text{-}Res}$ clauses are partitioned into two sets, P and R. Here, in a fair way, Γ inferences are performed on P and, with a *candidate set* of clauses, M, constructed from P, we saturate $M \cup R$ with Λ. Unsatisfiability in this system is witnessed by the unsatisfiability of P under Γ and satisfiability is witnessed by a saturated system without witnessing the unsatisfiability of P.

We present the inference system for $\Gamma + \Lambda$, discuss the soundness and provide a proof of its completeness. We also demonstrate how $\mathrm{Inst\text{-}Gen\text{-}Eq}$ and superposition can be combined into a single inference system, which we call $\mathrm{SIG\text{-}Sup}$, when viewed in the $\Gamma + \Lambda$ framework.

We have also begun work on an implementation of $\mathrm{SIG\text{-}Sup}$ which we call EVC3. EVC3 is our attempt to combine the strengths of existing software to implement the $\mathrm{SIG\text{-}Sup}$ inference system. EVC3 is a coupling of the SMT solver CVC3 [124] and the purely equational prover E [112]. Here we intend to use E for the superposition inferences and perform $\mathrm{Inst\text{-}Gen}$ inferences in a CVC3 quantification theory module that we have developed. Currently, EVC3 is a sound and complete implementation of $SIGRes$ but does not yet implement $SIGSup$.

Contributions to this work include all that is written in Chapter 5. This includes the formal description of the $\Gamma + \Lambda$ framework, the completeness proof, the formal description of $\mathrm{SIG\text{-}Sup}$, and our current work on EVC3. We note that it is our intent to submit a version of Chapter 5 for publication.

1.3 Outline of this Dissertation

In Chapter 2 we present the first-order predicate calculus and definitions for terms used throughout this dissertation. In Chapter 3 we discuss our method of using propositional encodings of tableaux proofs for first-order theorem proving and our implementation CHEWTPTP-SAT as well as our propositional encodings (modulo theories) and our implementation CHEWTPTP-SMT. In Chapter 4 we discuss our inference system SIG-Res and our implementation Spectrum and in Chapter 5 we discuss the $\Gamma + \Lambda$ framework, $\mathrm{SIG\text{-}Sup}$ and our initial work on an implementation called EVC3. A conclusion is given in Chapter 6.

Chapter 2

First-Order Logic

The alphabet for first-order logic consists of functions symbols, predicate symbols, variables, the quantifiers $\forall$ (universal) and $\exists$ (existential), the logical connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\rightarrow$ (implication), $\leftrightarrow$ (if and only if), and parentheses.

A *signature*, Σ, is a triple $(\mathcal{F}, \mathcal{P}, \alpha)$, where $\mathcal{F}$ is a set of function symbols, $\mathcal{P}$ is a set of predicate symbols, $\mathcal{F} \cap \mathcal{P} = \varnothing$ and $\alpha : \mathcal{F} \cup \mathcal{P} \rightarrow \mathbb{N}$ which defines the arity of the function and predicate symbols. We define constants as 0-ary function symbols. If Σ contains an n-ary function symbol f we say $f/n \in \Sigma$.

Terms are defined inductively as follows. Variables and constants are terms. If f is any n-ary function symbol and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term. We denote the (size) number of symbols in term t as $|t|$ and the number of occurrences of the variable x in term t as $|t|_x$.

If P is any n-ary predicate symbol and $t_1, \ldots, t_n$ are terms then $P(t_1, \ldots, t_n)$ is an *atomic formula* (atom). As a special case we allow the infix predicate symbol $\simeq$ and if t_i and t_j are terms then $t_i \simeq t_j$ is an atom. The negation of the equality atom $t_i \simeq t_j$ is written $t_i \not\simeq t_j$.

2.1 First-Order Formula

Well-formed formulas in first-order logic are constructed using the standard rules of formula construction given in Figure 2.1 [43].

(i) An atomic formula is a formula
(ii) If ϕ is a formula then $\neg\phi$ is a formula
(iii) If ϕ and ψ are formula then $(\phi \vee \psi)$ is a formula
(iv) If ϕ and ψ are formula then $(\phi \wedge \psi)$ is a formula
(v) If ϕ and ψ are formula then $(\phi \rightarrow \psi)$ is a formula
(vi) If ϕ and ψ are formula then $(\phi \leftrightarrow \psi)$ is a formula
(vii) If ϕ is a formula and x is a variable then $\exists x\phi$ is a formula
(viii) If ϕ is a formula and x is a variable then $\forall x\phi$ is a formula

Figure 2.1: Standard Rules for First-Order Formula Construction

A *literal* is either an atom (positive literal) or the negation of an atom (negative literal). Suppose L is a literal. The complement of L is denoted $\overline{L}$ and is defined as follows: if $L = A$ for some atom A then $\overline{L} = \neg A$ else if $L = \neg A$ for some atom A then $\overline{L} = A$. If L and K are literals with the same predicate symbol and opposite signs we say that L and K are complementary literals.

A *clause* is a disjunction of literals, however we often view a clause as a multiset of literals. A *ground clause* is a clause that contains no variables. We consider a formula to be a conjunction of clauses in conjunctive normal form and often view a formula as a set of clauses. We denote by $\varnothing$ the empty set and denote the empty clause by $\perp$. We define a *Horn clause* as a clause which contains at most one positive literal. A clause which contains only negative literals is called a *negative clause*.

2.2 Normal Forms

A formula is in *negation normal form* if it does not contain the logical connectives $\rightarrow$ and $\leftrightarrow$ and if all negations are applied to atoms. Any well-formed formula in the first-order logic can be transformed into a logically equivalent formula in negation normal form by

exhaustively applying the transformation rules in Figure 2.2 (in any order) [89]:

1.	$(A \rightarrow B)$	$\rightarrowtail (\neg A \vee B)$
2.	$(A \leftrightarrow B)$	$\rightarrowtail ((\neg A \vee B) \wedge (\neg B \vee A))$
3a.	$\neg(\forall v)A$	$\rightarrowtail (\exists v)\neg A$
3b.	$\neg(\exists v)A$	$\rightarrowtail (\forall v)\neg A)$
4.	$\neg\neg A$	$\rightarrowtail A$
5a.	$\neg(A \wedge B)$	$\rightarrowtail (\neg A \vee \neg B)$
5b	$\neg(A \vee B)$	$\rightarrowtail (\neg A \wedge \neg B)$

Figure 2.2: Negation Normal Form Transformation Rules

When a formula contains two or more variables that have the same name but are under the scope of different quantifiers we apply a renaming to the formula to give the variables under the scope of the innermost quantifier fresh variable names. For example we transform via renaming $(\forall x)P(x) \vee (\exists x)Q(x)$ to $(\forall x)P(x) \vee (\exists y)Q(y)$.

A formula with existential quantifiers can be transformed into an equisatisfiable formula without existential quantifiers via *Skolemization*. To Skolemize a formula we remove the existential by applying the following rules:

Let Φ be a formula containing a variable v that is under the scope of a existential quantifier.

- If v is not under the scope of a universal quantifier then Φ is equisatisfiable to $\Phi_{v \mapsto a}$[1] where a is a fresh constant.
- If v is under the scope of universal quantifiers $(\forall x_1)...(\forall x_j)$ then Φ is equisatisfiable to $\Phi_{v \mapsto f(x_1,...,x_j)}$ where f is a fresh function symbol.

A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses such that negations are applied only to atoms and all variables are universally quantified. A formula can be converted to an equisatisfiable formula in CNF by following the steps in Figure 2.3 [134].

[1] If Φ is a first-order formula and l and r are terms then $\Phi_{l \mapsto r}$ denotes the formula Φ with all occurrences of l replaced by r

1. Convert to negation normal form
2. Rename apart
3. Skolemize
4. Remove universal quantifiers
5. Distribute $\vee$s over $\wedge$s [2]

Figure 2.3: CNF Conversion Steps

2.3 Substitutions

A *substitution* is a mapping from variables to terms, almost everywhere the identity. We denote an application of a substitution σ to a clause C as $C\sigma$.

A clause C *subsumes* another clause D if there exists a substitution σ such that $C\sigma \subseteq D$. A *unifier* of two atoms L and K is a substitution σ such that $L\sigma = K\sigma$. The unification algorithm used in this work is given in Algorithm 1. In this algorithm, mgu stores the most general unifier and can be accessed in any number of ways. If such a unifier exists, we say that L and K are *unifiable*. A *most general unifier* of L and K, denoted $\mathrm{mgu}(L, K)$, is a unifier, σ, of L and K such that for every unifier, τ, of L and K, there exists some substitution ρ such that $\tau = \sigma\rho$ over the variables of L and K. A *renaming* is an injective substitution that maps variables to variables and we say that two literals are *variants* if there exists a renaming which unifies them.

A substitution that maps at least one variable of an expression E to a non-variable term is called a *proper instantiator* of E. We say that a clause C is a (proper) *instance* of clause C' if there exists some (proper instantiator) substitution σ such that $C = C'\sigma$. For a set of clauses S, we denote the set of all ground instances of the clauses in S as $Gr(S)$.

[2] We distribute $\vee$s over $\wedge$s using $A \vee (B \wedge C) \rightarrowtail (A \vee B) \wedge (A \vee C)$ and $(B \wedge C) \vee A \rightarrowtail (B \vee A) \wedge (C \vee A)$ for all formula A, B and C

Algorithm 1: Unify(Term p, Term q)

```
Let Q be an empty queue of pairs of terms
Let mgu be an empty list of pairs of terms
Push(Q, (p, q))
while Q is not empty do
    (s, t) := Pop(Q)
    if s is a term of the form f(s_1, ..., s_n) then
        if t is a variable then
            Push(Q, (t, s))
        else if t is a term with a different top symbol than f then
            Return false
        else
            Check the unifiability of the pairwise arguments of s and t
    else if s is a variable then
        if t is the same variable as s then
            Continue at top of while loop
        else if s occurs in t then
            Return false
        else
            Replace all instances of s in U and mgu with t and Push(mgu, (s, t))
    Return true
```

Chapter 3

Encoding First-Order Proofs in SAT and SMT

An impressive success recently in theorem proving has been the efficiency of SAT solving methods based on the DPLL method [29, 34]. The success of these methods seems to be based on the fact that the data structures are defined in advance and an exponential number of possibilities can be explored in polynomial space.

In [126] and [128] we try to take advantage of the efficiencies of SAT technology for first order theorem proving. The obvious idea is to try to incorporate a SAT solver and use it in such a way that it is not called often, because calling it too many times loses the advantages mentioned above.

In [126], we encode a proof of first order unsatisfiability with propositional clauses. We cannot, however, encode full first order unsatisfiability directly, since given a first-order formula Φ, a proof of the unsatisfiability of Φ contains an unbounded number of instances of the clauses in Φ, which cannot be encoded in propositional logic. We therefore chose to encode rigid first order unsatisfiability, which only requires us to encode that each clause appears at most a fixed number of times. Rigid unsatisfiability has been studied as early as [45, 50].

In order to encode a rigid proof, we need to decide what kind of proof should be encoded. We chose to encode a connection tableau proof because all of the clauses in the proof are instances of the original clauses whereas resolution proofs introduce clauses not contained in the original set. A set of first order clauses is rigidly unsatisfiable if and only if there exists a closed rigid connection tableau for that set of clauses [88]. Our method uses this fact and solves the unsatisfiability of a set of rigid clauses by encoding the existence of a rigid connection tableau in SAT.

The idea of the encoding is the following. We encode the existence of a clause as the root of the connection tableau. We encode the fact that every literal assigned to a non-leaf node is extended with a clause containing a complementary literal. Those things are easy to encode, and do not take up much space. There are three things which are more costly to encode.

First, we must encode the fact that two literals are complementary, in other words that their corresponding atoms are unifiable. For that, we basically have to encode a unification algorithm. In our encoding of unification, we leave out the occurs check, because it is expensive, and because it rarely occurs. We add a check for this after the SAT solver returns the truth assignment. If there really is an occurs check, we add a propositional clause and call the SAT solver again.

Second, the above encoding leaves open the possibility that the connection tableau is infinite. Therefore, we must encode the fact that the connection tableau is finite, i.e., that the connection tableau contains no cycle.

Third, we must encode the fact that every literal assigned to a leaf node is closed by a previous literal on its branch. Our encoding is simpler for the Horn case, because it is only necessary to close a literal with the previous one on the branch. For the non-Horn case, we must encode the fact that there is a complementary literal higher up in the tree. Since the same clause may occur on two different branches, and a literal on that clause may close with different literals on different branches, we may need to add more than one copy of a

clause in the rigid non-Horn case, because of the fact that the literal is closed differently. But we still try to get as much structure sharing in our tree as possible. Note that rigid Horn clause satisfiability is NP-complete, but Rigid non-Horn clause satisfiability is Σ_2^p-complete[1] [68]. So it is not surprising that a SAT solver cannot solve rigid non-Horn clause satisfiability directly.

Once we construct a propositional encoding of a rigid connection tableau proof for a first-order formula, we can utilize a SAT solver to establish the satisfiability of the encoding. Since SAT is a decidable problem, the SAT solver will return *rigidly* SATISFIABLE if a rigid connection tableau exits, and *rigidly* UNSATISFIABLE if a closed rigid connection tableau does not exist. If satisfiable, we can recover the rigid connection tableau proof from the truth assignment returned by the SAT solver.

Since we encode rigid proofs, the proof of unsatisfiability of a set of clauses may require augmenting the encoding using additional fresh variants of each clause. However, there are also applications which only require rigid proofs [122]. The SAT encodings are given in Section 3.2 and algorithms follow.

In [128] we encode a rigid connection tableaux proof of first order unsatisfiability in SMT rather than SAT. We expresses choices involved in building the tableau using propositional logic, and verification of unification and acyclicity using underlying theories. In our implementation called ChewTPTP-SMT we utilize the SMT solver called Yices.

Yices has a theory for recursive datatypes, which can be used to represent terms. A term can be defined by using constructors as function symbols. Each function symbol of arity n is defined by a constructor with n arguments. Constants are constructors with no arguments. Predicate symbols are viewed the same as function symbols. Variables are instances of types. Then unification is represented as equality of types. We represent acyclicity using linear arithmetic. Consider a graph $G = (V, E)$. If an edge (u, v) exists in E, then we assert an inequality $x_u < x_v$ for some real numbers x_u and x_v. Then G

[1] $\Sigma_2^p := NP^{\Sigma_1^p} := NP^{NP}$ is the set of decision problems solvable by a Turing machine in NP that is augmented by an oracle in NP

is acyclic if and only if the set of inequalities is consistent. The encodings in SMT can be found in Section 3.3 with a description of our implementation ChewTPTP-SMT and experimental results to follow.

3.1 Preliminaries

3.1.1 Propositional Logic

The alphabet for propositional logic formula consists of propositional variables and the logical connectives $\vee$ (disjunction), $\wedge$ (conjunction), $\neg$ (negation) and parentheses. As with first order logic, we will consider propositional logic formulas to be in CNF and conform to the standard rules for constructing valid propositional logic formulas[43].

3.1.2 Connection Tableaux

We define rigid clausal tableau as follows.

Definition 1 *Rigid clausal tableaux are trees with nodes labeled with literals, branches labeled either open or closed, and edges labeled with zero or more assignments. Rigid clausal tableaux are inductively defined as follows. Let $S = \{C_1...C_n\}$ be a set of clauses. If T is a tree consisting of a single unlabeled node (the root node) N then T is a rigid clausal tableau for S. The branch consisting of only the root node N is open. If N is a leaf node on an open branch B in the rigid clausal tableaux T for S and one of the following inference rules are applied to T then the resulting tree is a rigid clausal tableaux for S.*

(Expansion rule) Let $C_k = L_{k1} \vee ... \vee L_{ki}$ be a clause in S. Construct a new tableaux T' by adding i nodes to N and labeling them L_{k1} through L_{ki}. Label each of the i branches open.

(Closure rule) Suppose L_{ij} is the literal at N and for some predecessor node M with literal L_{pq} there exists some most general unifier σ such that $L_{ij}\sigma = \neg L_{pq}\sigma$ and the assign-

ments of σ are consistent with the assignments of T. Construct T' from T by labeling the edge from L_{pq} to L_{ij} with the assignments used in the unification and by closing the branch of N.

We call the clause which is added to the root node the *start clause* and we say that a clause is *in* a tableau if the clause was used in an application of the expansion rule.

Definition 2 *A clausal tableaux is connected if each clause (except the start clause) in the tableaux contains some literal which is unifiable with the negation of its predecessor [93].*

Definition 3 *(Extension Rule) Let N be a node in the tableau T and let C_k be a clause in S such that there exists a literal L_{ki} in C_k which is unifiable with the negation of N. Apply the expansion rule with C_k and immediately apply the closure rule with L_{ki}.*

The calculus for connection tableaux (or model elimination tableau [93]) consists of the expansion rule (for the start clause only), the closure rule, and the extension rule. We call a tableau *closed* if each leaf node has been closed by an application of the closure rule.

By [93] we can require that the start clause be a negative clause since there exists a negative clause in any minimally satisfiable set.

3.1.3 Rigid Unsatisfiability

Unless otherwise stated, we let F be a set of first order logic formulas. The main problem in Automated Theorem Proving is to determine if the set of hypotheses in F implies the conclusion in F. For our purposes we assume that all formula in a problem are in CNF and the conclusion is negated. Therefore we seek to determine if F is (equivalently) unsatisfiable, i.e. there does not exist a model for F. The problem of rigid unsatisfiability of F seeks to determine whether there exists a ground instance of F which is unsatisfiable.

A result of Tableau Theory is the completeness and soundness of closed (rigid) connection tableaux.

Theorem 1 *There exists a closed (rigid) connection tableau for F iff F is (rigidly) unsatisfiable[88].*

3.2 Encoding in SAT

Our method to determine the rigid satisfiability of F generates a set of propositional logic clauses for F which encodes a closed rigid connection tableau for F. We provide two encoding, the first for problems which contain only Horn clauses and the second for those containing non-Horn clauses. Given F, we give unique symbols to each of the clauses in F and each of the literals in each clause. We represent clause i by C_i. We represent the j^{th} literal in clause i by L_{ij} (which is used to label the tableaux). Note that as multiple copies of a clause may appear in a rigid connection tableau, multiple nodes may have the same literal label. And whereas the same literal may appear in distinct clauses, they are identified with different labels. We denote A_{ij} to be the atom of L_{ij}. Therefore L_{ij} is either of the form A_{ij} or $\neg A_{ij}$.

3.2.1 Encoding for Horn Clauses

We define the variables c_m, l_{mn}, e_{mnq}, u_k, p_{mq} as follows: Define $c_m = T$ iff C_m appears in the tableau. Define $l_{mn} = T$ iff L_{mn} is an internal node in the tableau. Define $e_{mnq} = T$ iff C_q is an extension of L_{mn}. Define $u_\tau = T$ iff τ is an assignment implied by the substitutions used in the closure rules. Define $p_{mq} = T$ iff there exists a path from a literal in C_m to C_q.

Below we list the set of clauses that we generate and provide their meaning.

At least one clause containing only negative literals appears in the tableau:

$$\bigvee_{C_m \text{ is a negative clause}} c_m \tag{3.1}$$

If C_m appears in the tableau and L_{mn} is a negative literal then L_{mn} is an internal node

in the tableau:

$$\neg c_m \vee l_{mn} \tag{3.2}$$

If L_{mn} is an internal node in the tableau then for some q_j, C_{q_j} is an extension of L_{mn}:

$$\neg l_{mn} \vee e_{mnq_1} \vee ... \vee e_{mnq_k} \tag{3.3}$$

where $\{C_{q_1}...C_{q_k}\}$ represents the set of all clauses whose positive literals are unifiable with L_{mn}

If C_q is an extension of L_{mn} then C_q exists in the tableau:

$$\neg e_{mnq} \vee c_q \tag{3.4}$$

If C_q is an extension of L_{mn} and τ is an assignment of the mgu used to unify A_{qr} with A_{mn} then τ is implied by the mgu:

$$\neg e_{mnq} \vee u_\tau \text{ where } \tau \in mgu(A_{mn}, A_{qr}) \tag{3.5}$$

If for two assignments $x = s$ and $x = t$ there does not exist a mgu θ such that $s\theta = t\theta$ then both assignments can not be true:

$$\neg u_{x=s} \vee \neg u_{x=t} \text{ where } s \text{ and } t \text{ are not unifiable} \tag{3.6}$$

If $x = s$, $x = t$, $\sigma = mgu(s, t)$ and $y = r \in \sigma$ then $y = r$:

$$\neg u_{x=s} \vee \neg u_{x=t} \vee u_{y=r} \text{ where } y = r \in mgu(s, t) \tag{3.7}$$

If C_q is an extension of L_{mn} then there is a path from C_m to C_q:

$$\neg e_{mnq} \vee p_{mq} \tag{3.8}$$

Transitivity of the path relation:

$$\neg p_{mq} \vee \neg p_{qs} \vee p_{ms} \tag{3.9}$$

There are no cycles in the tableau:

$$\neg p_{mm} \tag{3.10}$$

3.2.2 Encoding for Non-Horn Clauses

For non-Horn problems we use an alternative set of variables and generate a different set of clauses.

We define the variables s_m, c_{mn}, l_{mn}, e_{mnqj}, u_k, p_{mq}, and q_{mnij} as follows. Define $s_m = T$ iff C_m is the start clause. Define $c_{mn} = T$ iff C_m appears in the tableau and L_{mn} is used to close its parent. Define $l_{mn} = T$ iff L_{mn} is a node in the tableau and is not a leaf node created by an application of the closure rule. Define $e_{mnqj} = T$ iff C_q is an extension of L_{mn} and L_{mn} is used to close L_{qi}. Define $u_\tau = T$ iff τ is an assignment implied by the unifiers used in the applications of the closure rules. Define $o_{ijkl} = T$ iff L_{kl} is used to close L_{ij}. Define $p_{mq} = T$ iff there exists a path from a literal in C_m to C_q. Define $q_{mnij} = T$ iff L_{mn} is a leaf and L_{mn} is closed by a literal between the root node and L_{ij}.

The clauses are as follows.

There exists a start clause in the tableau which only contains negative literals:

$$\bigvee_{s_m \text{ is a negative clause}} s_m \tag{3.11}$$

If C_m is the start clause in the tableau then each literal L_{mn} of C_m is in the tableau:

$$\neg s_m \vee l_{mn} \tag{3.12}$$

If C_i appears in the tableau and L_{ij} is the complement of a literal in its parent then all other literals of C_i are in the tableau:

$$\neg c_{ij} \vee l_{ik} \text{ where } j \neq k \tag{3.13}$$

If L_{ij} exists in the tableau and is not a leaf node created by an application of the closure rule then either L_{ij} is closed by a literal between the root and L_{ij} or there is an extension of L_{ij}:

$$\neg l_{ij} \vee q_{ijij} \bigvee_{k,l} e_{ijkl} \tag{3.14}$$

If L_{ij} is extended with C_k then C_k is in the tableau and some L_{kl} of C_k is closed by L_{ij}:

$$\neg e_{ijkl} \vee c_{kl} \tag{3.15}$$

If clause C_m is an extension of L_{ij} and τ is an assignment of the mgu used to unify A_{ml} with A_{ij} then τ is true:

$$\neg e_{ijml} \vee u_\tau \text{ where } \tau \in mgu(A_{ml}, A_{ij}) \tag{3.16}$$

If for two assignments $x = s$ and $x = t$ there does not exist a mgu θ such that $s\theta = t\theta$ then both assignments can not be true:

$$\neg u_{x=s} \vee \neg u_{x=t} \text{ where } s \text{ and } t \text{ are not unifiable} \tag{3.17}$$

If $x = s$, $x = t$, $\sigma = mgu(s, t)$ and $y = r \in \sigma$ then $y = r$:

$$\neg u_{x=s} \vee \neg u_{x=t} \vee u_{y=r} \text{ where } y = r \in mgu(s, t) \tag{3.18}$$

If L_{ij} is used to close L_{kl} then their atoms must be unifiable by some unifier σ, hence each assignment of σ is true:

$$\neg o_{ijkl} \vee u_{\tau} \text{ where } \tau \in mgu(A_{ij}, A_{kl}) \tag{3.19}$$

If L_{ij} has the same sign as L_{kl} or their respective atoms are not unifiable then L_{ij} is not used to close L_{kl}:

$$\neg o_{ijkl} \text{ where } L_{ij} \text{ and } L_{kl} \text{ have the same sign or } A_{ij} \text{ and } A_{kl} \text{ are not unifiable} \tag{3.20}$$

If leaf L_{ij} is closed by a literal between the root and L_{kl} and clause C_k is an extension of L_{mn} then L_{ij} is closed by some literal between the root and L_{mn}:

$$\neg q_{ijkl} \vee \neg e_{mnkl} \vee o_{ijmn} \vee q_{ijmn} \tag{3.21}$$

If C_k is an extension of L_{ij} then there is a path from clause C_i to clause C_k:

$$\neg e_{ijkl} \vee p_{ik} \tag{3.22}$$

Transitivity for paths:

$$\neg p_{ij} \vee \neg p_{jk} \vee p_{ik} \tag{3.23}$$

There are no cycles in the tableau:

$$\neg p_{ii} \tag{3.24}$$

If C_i is the start clause then there are no extensions into any of the literals in C_i:

$$\neg s_i \vee \neg e_{klij} \tag{3.25}$$

If C_i is the start clause and L_{mn} is a leaf which is closed by a literal between the root node and L_{ij}, then L_{mn} must be closed with L_{ij}:

$$\neg s_i \vee \neg q_{mnij} \vee o_{mnij} \tag{3.26}$$

3.2.3 Tableau Encoding Algorithms

We provide three algorithms, each with subtle differences. The first algorithm HTE attempts to find a rigid proof and takes as an argument a problem containing only Horn clauses. The second, $NHTE$, also attempts to find a rigid proof and takes as an argument a non-Horn problem. The last algorithm, $NRTE$, seeks to finds a non-rigid proof and takes either a Horn or non-Horn problem as an argument.

The rigid algorithm for non-Horn problems may require additional copies of the clauses in F in order to generate a proof for F and the non-rigid algorithm may also require additional instances of clauses. In the case of the former, copies of clauses in F are added to the set of problem clauses. The number of copies required can be bounded by k^n where n is the number of clauses in F and k is the maximum number of literals in any clause in F. In the case of the non-rigid algorithm, new instances of clauses in F which are standardized apart are added to the problem clauses.

Each algorithm initially enters a while loop. While in the loop the set of clauses S, which encode the closed rigid connection tableau, is given to an external SAT solver. The SAT solver returns satisfiable or unsatisfiable and if the set of clauses is satisfiable, the SAT solver returns a model M. If the SAT solver returns satisfiable we check if the assignments

which are assigned true in M are consistent. If not, we add additional clauses to S to resolve these inconsistencies and call the SAT solver again. If the algorithm determines that S is satisfiable and the assignments which are assigned true are consistent, the algorithm returns an indication that F is rigidly unsatisfiable.

The function Unify-Substitutions takes as an argument the model M generated by the SAT solver and generates additional clauses to rectify inconsistencies in the assignments used in the proof. The only inconsistency that can occur among assignments is due to cycles. For example, $\{x_1 = f(x_2), x_2 = f(x_3), x_3 = f(x_1)\}$. If a cycle is found, a clause is created which prevents the conflict. These clauses are added to the original set of clauses generated by the algorithm which are again checked by the SAT solver.

Algorithm 2: Rigid Algorithm For Horn Problems (HTE)

input : F, a set of FO formula in conjunctive normal form
output: RIGIDLY SATISFIABLE or RIGIDLY UNSATISFIABLE

Generate S, the encodings for F;
while $true$ **do**
    result := `SAT-Solver(`$S \cup S'$`);`
    if result == SATISFIABLE *and the model* M *is consistent* **then**
        return RIGIDLY UNSATISFIABLE;
    else if result == SATISFIABLE **then**
        S' := `Unify-Substitution(`M`);`
    else
        return RIGIDLY SATISFIABLE;

3.2.4 Completeness and Soundness Theorems for HTE

In the following proofs we refer to the sets of clauses generated by HTE by the enumeration given in Section 3.2.1.

Theorem 2 *(Completeness) Let F be a set of first order logic Horn clauses. If F is rigidly unsatisfiable, then HTE will return RIGIDLY UNSATISFIABLE.*

Algorithm 3: Rigid Algorithm For Non-Horn Problems (NHTE)

input : F, a multi-set of FO formula in conjunctive normal form
output: RIGIDLY UNSATISFIABLE

```
F' := F;
S' := ∅;
while true do
    Generate S, the encoding for F';
    result := SAT-Solver(S ∪ S');
    if result == SATISFIABLE and the model M is consistent then
        return RIGIDLY UNSATISFIABLE;
    else if result == SATISFIABLE then
        S' := Unify-Substitution(M);
    else
        F' = F' ∪ F;
```

Algorithm 4: Non-Rigid Algorithm (NRTE)

input : F, a set of FO formula in conjunctive normal form
output: RIGIDLY UNSATISFIABLE

```
F' := F;
S' := ∅;
while true do
    Generate S, the encoding for F';
    result := SAT-Solver(S ∪ S');
    if result == SATISFIABLE and the model M is consistent then
        return RIGIDLY UNSATISFIABLE;
    else if result == SATISFIABLE then
        S' := Unify-Substitution(M);
    else
        Generate set of variants, A, of F. F' = F' ∪ A;
```

Proof Assume F is rigidly unsatisfiable and let S be the set of clauses for F generated by HTE. As F is rigidly unsatisfiable then by Theorem 1 there exists a closed rigid connection tableaux T. It also follows that the start node of T contains only negative literals. From T we will construct a map from the variables in S to {true, false} so that S is satisfiable.

If C_m appears in the tableau set $c_m = true$ otherwise set $c_m = false$. If L_{mn} is an internal node in the tableau set $l_{mn} = true$ otherwise set $l_{mn} = false$. If C_q is an extension of L_{mn} set $e_{mnq} = true$ otherwise set $e_{mnq} = false$. If τ is an assignment implied by the unifiers used applications of the closure rule set $u_\tau = true$ otherwise set $u_\tau = false$ and if there exists a path from C_m to C_q set $p_{mq} = true$ otherwise set $p_{mq} = false$.

As T has a start node containing only negative literals, there exists a variable in Set 1 which is true. Thus Set 1 of S is satisfiable.

As T is a connection tableau and each extension of T closes the branch containing the positive literal of a clause, and since each clause contains at most one positive literal, then each negative literal in T is an internal node. Hence each variable representing a clause in T is true iff its negative literal variables are also true. Thus Set 2 is satisfiable.

Since each negative literal in T must be extended it follows that each variable representing a negative literal in T is true iff the variable representing its extension is true. Therefore Set 3 is satisfiable. Furthermore since each extension of T extends a literal to all the literals in a clause, an extension variable is true iff the clause variable associated with the extension is true. Thus Set 4 is satisfiable.

Since each extension in T unifies complementary literals, it follows that an extension variable is true iff each of the variables representing the assignments in the unifier used in the unification of the complementary literals are true. Hence Set 5 is satisfiable. It also follows by the consistency of T that inconsistent assignments can not both be true, thus for each pair of variables representing inconsistent assignments we have one is true iff the other is false. Hence Set 6 is satisfiable. In addition if two assignments map the same variable to unifiable terms s and t then the assignments used in the unification of s and t

must be true. Therefore Set 7 is satisfiable.

Now as there exists paths between literals and clauses via extensions in T, if a variable representing an extension is true then the variable representing the path is true. Thus Set 8 is satisfiable. And since the paths in T have a transitive relation and no cycles exist in T, Sets 9 and 10 are satisfiable respectively.

Therefore since each of the sets of clauses in S are satisfiable, then the SAT solver called in HTE returns a satisfiable model with consistent assignments, hence HTE returns RIGIDLY UNSATISFIABLE. ∎

Theorem 3 *(Soundness) If HTE on F returns RIGIDLY UNSATISFIABLE then F is rigidly unsatisfiable.*

Our proof of soundness uses the satisfiability map produced by HTE to construct a tableau for F.

Proof Suppose HTE on F returns RIGIDLY UNSATISFIABLE. Then there exists a set of clauses S generated by HTE and a model M for which S is satisfiable. Furthermore the set of assignment variables that are true in M correspond to a consistent set of assignments. We construct a closed rigid connection tableau T for F using M and S as follows.

Since S is satisfiable the clause $C = c_1 \vee ... \vee c_n$ in Set 1 of S, is satisfiable. Since C is satisfiable at least one of the variables in C are assigned true. Let c_m where $m \in [1..n]$ be a variable of C such that $c_m = true$. We begin constructing T by setting C_m as the start clause of T. [2]

Now as $c_m = true$ and Set 2 is satisfiable, each of the variables corresponding to the literals in C_m are true. Thus for each literal L_{mn} in C_m we create a node directly off the root and label it L_{mn}.

Let L_{mn} be a literal in C_m. Now as l_{mn} is true and Set 3 is satisfiable there exists some variable e_{mnq_i} which is true and as Set 4 is satisfiable $e_{mnq_i} = true$ implies $c_{q_i} = true$. We

[2] It may be the case that more than one variable of C is assigned true. This corresponds to the fact that there may be more than one closed rigid connection tableau for F.

therefore expand the node labeled L_{mn} in T with clause C_{q_i}. We continue this process until all literal, clause, and extention variables which are assigned true have been addressed. By the satisfiability of Sets $2-4$, T is closed.

Now let e_{mnq_i} be a variable in M which is set to true. Since Set 5 is satisfiable, e_{mnq_i} implies that a set of assignments are true. We label the edge from L_{mn} to the positive literal in C_{q_i} with these assignments. Since each extension unifies adjacent complementary literals and the assignments in M are consistent, T is connected and consistent.

The satisfiability of Sets $8-10$ ensure that there are no cycles in T, hence T is a tree. It follows then that T is a closed connection tableau. Since each clause in T is in F, T is a closed rigid connection tableau for F. Thus by the soundness theorem for closed rigid connection tableaux, F is rigidly unsatisfiable. ∎

3.2.5 Completeness and Soundness Theorems for NHTE

Here we provide the completeness theorem of NHTE which takes as input non-Horn problems. In the proofs, we refer to the sets of clauses generated by NHTE by the enumeration given in Section 3.2.2.

Theorem 4 *(Completeness) Let F be a set of first order clauses. If F is rigidly unsatisfiable, then NHTE will return RIGIDLY UNSATISFIABLE.*

Proof Assume F is rigidly unsatisfiable and let S be the set of clauses for F generated by NHTE. By Theorem 1, as F is unsatisfiable, there exists a closed rigid connection tableaux T for F. It also follows that the start node of T contains only negative literals. Let S be the set of clauses generated by NHTE. Given T we will construct a map from the variables in S to {true, false} so that S is satisfiable.

Set $s_m = T$ iff C_m is the start clause. Set $c_{mn} = T$ iff C_m appears in the tableau and L_{mn} is closed by an application of the extension rule. Set $l_{mn} = T$ iff L_{mn} is a node in the tableau but is not closed by an application of the extension rule. Set $e_{mnqj} = T$ iff C_q

is an extension of L_{mn} and L_{qj} closes L_{mn}. Set $u_\tau = T$ iff τ is a assignment implied by substitutions used in the closure rules. Set $o_{ijkl} = T$ iff L_{kl} is used to close L_{ij} but not during an application of the expansion rule. Set $p_{mq} = T$ iff there exists a path from a literal in C_m to C_q. Set $q_{mnij} = T$ iff L_{mn} is a leaf and is closed by a literal between the root node and L_{ij}.

As T has a start node containing only negative literals, there exists a variable in Set 11 which is true, thus Set 11 of S is satisfiable. Since each of the literals in the start clause are in T and are not closed by an application of the expansion rule then their respective variables are true, therefore Set 12 is satisfiable.

Now as each clause in T (except for the start clause) is the result of an expansion rule, and only one literal in each clause is closed in the process of using the expansion rule, all the other literals are in the tableau but are not closed by an application of the expansion rule. Hence Set 13 of S is satisfiable.

Suppose L_{ij} is in T such that L_{ij} is not closed by an application of the expansion rule. Then either L_{ij} is extended or L_{ij} has been closed by a complementary literal on its path. It follows that Set 14 is satisfiable.

Since each extension in T adds a clause to T, Set 15 is satisfiable. Since each extension in T unifies complementary literals, it follows that an extension variable is true iff each of the variables representing the assignments in the unifier used in the unification of the complementary literals are true. Hence Set 16 is satisfiable. It also follows by the consistency of T that inconsistent assignments cannot both be true, thus for each pair of variables representing inconsistent assignments, one is true iff the other is false. Hence Set 17 is satisfiable. In addition if two assignments map the same variable to unifiable terms s and t then the assignments used in the unification of s and t must be true. Therefore Set 18 is satisfiable.

As each pair of literals which are used in a non-extension closure are complements, if a variable representing the non-extension closure between two literals is true then the

variables representing the assignments implied by unification of their atoms are true. Hence Set 19 is satisfiable. Since no two literals with have the same sign or which have atoms that are not unifiable cannot be used in a non-extension closure, Set 20 is satisfiable.

Suppose L_{ij} is a leaf and is closed by a literal between the root and L_{kl}. If the clause containing L_{kl} is an extension of some node L_{mn} then either L_{mn} is a complement of L_{ij} or L_{ij} is closed by a literal between the root node and L_{mn}. It follows that Set 21 is satisfiable.

Now as there exists paths between literals and clauses via extensions in T, if a variable representing an extension is true then the variable representing the path is true. Thus Set 22 is satisfiable. And since the paths in T have a transitive relation and no cycles exist in T, Sets 23 and 24 are satisfiable respectively.

As the start clause has no expansions into it, Set 25 is satisfiable. And since if a leaf, say L_{ij} in T is closed by a non-extension closure by a literal between the root and L_{mn} of the start clause, since there are not literals between the root and the literals of the start clause, then L_{ij} must be closed by L_{mn}. Hence Set 26 is satisfiable.

Therefore as each of the sets of clauses in S are satisfiable, then the SAT solver called in NHTE returns SATISFIABLE. It follows that as T is a tableau the assignments implied by the closure rule are consistent. Hence, NHTE returns RIGIDLY UNSATISFIABLE.

Theorem 5 *(Soundness) If NHTE on F returns RIGIDLY UNSATISFIABLE then F is rigidly unsatisfiable.*

3.2.6 ChewTPTP-SAT

We have implemented our tableau encoding method in a command line program written in C++ called ChewTPTP-SAT. The default options assume the input file is in TPTP CNF format [79]. By default the program assumes the input problem is non-Horn and uses the non-Horn algorithm with one instance of the clauses in the input file. The user may specify alternate settings by including the following flags. The flag `-h` indicates the problem is Horn, `-r` specifies the user wishes the program to run one of the rigid algorithms, `-i`

allows the user to input the number of instances of the problem to use, and `-p` instructs the program to print a proof. Other options are provided to control input and output.

The program initially parses the input file and constructs a data structure to hold the clauses in memory. The program then constructs the sets of clauses defined in section 3.2.1 or section 3.2.2. While generating the clauses, a data structure is kept which maps each variable to a unique integer. We use the integers to format the clauses in a MiniSat [108] readable format. ChewTPTP-SAT then forks a process and invokes MiniSat on the set of generated clauses and MiniSat determines the satisfiability of the set. When MiniSat returns, we inspect the file output by MiniSat. If the file contains an indication of satisfiability we check that the substitutions are unifiable and if so, we use the model provided by MiniSat to construct a proof. If MiniSat returns back an indication of unsatisfiable, the program returns SATISFIABLE in the rigid Horn case, and may add additional clauses and repeat the process in the other cases.

Experimental Results

Preliminary results on 1365 Horn and non-Horn CNF problems without equality in the TPTP Library show that 221 of them have rigid proofs requiring a single instance. We have found that ChewTPTP-SAT was able to solve some problems which many theorem provers could not within a 600 second time limit, e.g. the non-Horn problems ANA003-4.p and ANA004-4.p. And although we have not tested the library extensively by adding additional instances, ChewTPTP-SAT was successful solving non-rigid problems that others were unable, e.g. ANA003-2 was proved with 2 instances in less than 5 seconds.

Below in Table 3.2.6 are some statistics on the problems mentioned above and a few other problems. The first column identifies the name of the problem in the TPTP library and the second column identifies whether or not the problem is Horn. The third column identifies the number of instances that were required to prove the problem. The fourth column gives the number of seconds ChewTPTP-SAT took to generate the tableau encoding(s)

Table 3.1: Statistics on Selected Problems

Name	Horn	Instances	Clause Gen (sec)	MiniSat (sec)	Clauses	Variables
ALG002-1	N	2	1.2	65.93	411020	13844
ANA003-2	Y	2	.1	4.88	183821	7238
ANA003-4	N	1	1.1	.06	34774	2616
ANA004-4	N	1	1.61	.3	44142	3160
COL121-2	Y	1	1.35	.16	47725	2322
GRP029-2	Y	1	.08	1.41	241272	7943
PUZ031-1	N	1	.24	.71	662145	14672

and the fifth column gives the total time (in seconds) that MiniSat ran on the problem. The sixth and last columns give the number of clauses and variables respectively that were input to MiniSat when MiniSat returned SATISFIABLE.

In our ChewTPTP-SAT implementation, some problems generate large encodings for MiniSat to solve and Minisat usually solves them very quickly. The implementation shows promise given that it can solve some problems quickly that many other theorem provers cannot solve. Obviously it will perform best on problems that do not need many instances of the clauses. From our results, it appears that more than 15% of the problems without equality in the TPTP library are rigidly unsatisfiable, requiring only one instance of each clause. Further investigation, however, needs to be done to identify which class of problems our method does better on.

Implementation Status

ChewTPTP-SAT, in its current state, is a sound and complete theorem prover for first-order logic without equality. Though we have been successful in finding problems that ChewTPTP-SAT can solve and others theorem provers can not, the implementation is not a overall competitive solution to the first-order validity problem. For example, when submitted into the CASC-J4, the 2008 CADE Automated Theorem Proving System Competition, [131], ChewTPTP-SAT solved 6 of 100 problems in the CNF division whereas Vampire, the winner of the division, solved 93 problems.

We have however identified ways to improve our implementation which we believe will make the implementation more competitive. One future modification to the implementation deals with the manner in which the SAT solver is called. Currently, each time the encoding is modified, the entire encoding is sent to a new instance of the SAT solver. This results in the SAT solver searching the same space repeatedly. To avoid this repeated work, a single instance of the SAT solver can be kept in memory while the encoding is augmented. New clauses added to the encoding can be sent to the SAT solver and when all new clauses are added, a new satisfiability query can be made. Eliminating restarts should provide a significant improvement in performance.

3.3 Encoding in SMT

Our second method to determine the rigid unsatisfiability of F generates a set S of propositional logic clauses modulo the theories of unification and arithmetic for F which encodes a rigid closed connection tableau for F and tests the satisfiability of S with a SMT solver.

We provide two encodings, the first for problems containing only Horn clauses and the second for those containing non-Horn clauses. Given F we enumerate each of the clauses in F and each of the literals in each clause. We denote clause i by C_i and denote the j^{th} literal in clause i by L_{ij}. We denote A_{ij} to be the atom of L_{ij}. Therefore L_{ij} is either of the form A_{ij} or $\neg A_{ij}$.

3.3.1 Encoding for Horn Clauses

Let F be a set of first order logic formulas.

We define a set of propositional variables c_m, l_{mn}, e_{mnq}, disjoint from the symbols in F, as follows: Define $c_m = T$ iff C_m appears in the tableau. Define $l_{mn} = T$ iff L_{mn} is an internal node in the tableau. Define $e_{mnq} = T$ iff C_q is an extension of L_{mn}. For each pair of clauses C_i and C_j we define $x_i < x_j = T$ (where x_i and x_j do not exist in F)

iff there exists a path from C_i to C_j. For each pair of atoms A_i and A_j in F, we define $(A_i = A_j) = T$ iff A_i and A_j are the two atoms involved in an application of the closure rule.

Below we list the set of clauses that we generate and provide their meaning.

At least one clause containing only negative literals appears in the tableau:

$$\bigvee_{C_m \text{ is a negative clause}} c_m \tag{3.27}$$

If C_m appears in the tableau and L_{mn} is a negative literal then L_{mn} is an internal node in the tableau:

$$c_m \Rightarrow l_{mn} \tag{3.28}$$

If L_{mn} is an internal node in the tableau then for some q_j, C_{q_j} is an extension of L_{mn}:

$$l_{mn} \Rightarrow (e_{mnq_1} \vee ... \vee e_{mnq_k}) \tag{3.29}$$

where $\{C_{q_1}...C_{q_k}\}$ represent the set of all clauses whose positive literals are unifiable with L_{mn}

If C_q is an extension of L_{mn} then C_q exists in the tableau:

$$e_{mnq} \Rightarrow c_q \tag{3.30}$$

If C_q is an extension of L_{mn} and L_{qr} is the positive literal in C_q then A_{mn} and A_{qr} are unifiable:

$$e_{mnq} \Rightarrow (A_{mn} = A_{qr}) \tag{3.31}$$

If C_q is an extension of L_{mn} then there is a path from C_m to C_q:

$$e_{mnq} \Rightarrow (x_m < x_q) \tag{3.32}$$

The encoding is satisfiable if and only if the original set of first order Horn clauses is rigidly unsatisfiable. We encode non-rigid unsatisfiability by continually adding new instances of each clause, renamed apart.

3.3.2 Encoding for Non-Horn Clauses

For non-Horn problems we use a different set of variables and generate a different set of clauses.

We define the variables, disjoint from the symbols in F, s_m, c_{mn}, l_{mn}, e_{mnqj}, o_{ijkl} and q_{mnij} as follows: Define $s_m = T$ iff C_m is the start clause. Define $c_{mn} = T$ iff C_m appears in the tableau and L_{mn} is complementary to its parent. Define $l_{mn} = T$ iff L_{mn} is a node in the tableau and is not a leaf node created by an application of the extension rule. Define $e_{mnqj} = T$ iff C_q is an extension of L_{mn} and L_{qj} is the complement of L_{mn}. Define $o_{ijkl} = T$ iff L_{ij} and L_{kl} are a pair of literals used in a closure but not by the extension rule. If a path to a node N contains the complement of N, then we say that the path is closed. Define $q_{mnij} = T$ iff L_{mn} is a leaf and L_{ij} is a node on a path from the root node to L_{mn} and every path from the root to L_{ij} contains a complement of L_{mn}. For each pair of clauses C_i and C_j we define $x_i < x_j = T$ (where x_i and x_j do not exist in F) iff there exists a path from C_i to C_j. For each pair of atoms A_i and A_j in F, we define $(A_i = A_j) = T$ iff A_i and A_j are the two atoms involved in an application of the closure rule.

The clauses are as follows.

There exists a start clause in the tableau which only contains negative literals:

$$\bigvee_{s_m \text{ is a negative clause}} s_m \tag{3.33}$$

If C_m is the start clause in the tableau then each literal L_{mn} of C_m is in the tableau:

$$s_m \Rightarrow l_{mn} \tag{3.34}$$

If C_i appears in the tableau and L_{ij} is the complement of a literal in its parent then all other literals of C_i are in the tableau:

$$c_{ij} \Rightarrow l_{ik} \text{ where } j \neq k \tag{3.35}$$

If L_{ij} exists in the tableau and is not a leaf node created by an application of the closure rule then either every branch ending at L_{ij} is closed or there is an extension of L_{ij}:

$$l_{ij} \Rightarrow (q_{ijij} \vee (\bigvee_{k,l} e_{ijkl})) \tag{3.36}$$

If L_{ij} is extended with C_k then C_k is in the tableau and some L_{kl} of C_k is the complement of L_{ij}:

$$e_{ijkl} \Rightarrow c_{kl} \tag{3.37}$$

If clause C_m is an extension of L_{ij} and literals L_{ij} and L_{ml} are complements then A_{ij} and A_{ml} are unifiable.

$$e_{ijml} \Rightarrow (A_{ij} = A_{ml}) \tag{3.38}$$

If L_{ij} and L_{kl} are a pair used in a closure then they must be unifiable:

$$o_{ijkl} \Rightarrow (A_{ij} = A_{kl}) \tag{3.39}$$

If L_{ij} has the same sign as L_{kl} or their respective atoms are not unifiable then they are not complements:

$$\neg o_{ijkl} \text{ where } L_{ij} \text{ and } L_{kl} \text{ are not unifiable} \tag{3.40}$$

If every path through L_{kl} to leaf L_{ij} is closed and C_k is an extension of L_{mn} then either L_{ij} is a complement of L_{mn} or every path through L_{mn} to L_{ij} is closed:

$$q_{ijkl} \Rightarrow (e_{mnkp} \Rightarrow (o_{ijmn} \vee q_{ijmn})) \tag{3.41}$$

If C_k is an extension of L_{ij} then there is a path from clause C_i to clause C_k:

$$e_{ijkl} \Rightarrow (x_i < x_k) \tag{3.42}$$

If C_i is the start clause then there are no inferences into any of the literals in C_i:

$$s_i \Rightarrow \neg e_{klij} \tag{3.43}$$

If C_i is the start clause, L_{mn} is a leaf, and all paths that traverse L_{ij} to L_{mn} are closed, then L_{ij} and L_{mn} are complementary:

$$s_i \Rightarrow (q_{mnij} \Rightarrow o_{mnij}) \tag{3.44}$$

We represent our tableau as a DAG, so there is some structure sharing. But even with the structure sharing, a non-Horn clause tableau may need more than one instance of the same clause. Rigid unsatisfiability could be determined by continually adding identical instances of a clause. Non-Horn encoding could also be extended to the non-rigid case in the same way as the Horn encoding.

3.3.3 ChewTPTP-SMT

We have implemented our tableau encoding in our theorem prover ChewTPTP-SMT, which is an extension of ChewTPTP-SAT[126]. In ChewTPTP-SAT, instead of using theories, we

encoded the consistency of the unifiers and the acyclicity of the tableau with additional propositional clauses. To encode the consistency of the unifiers, we encoded the equations that would be created if a unification algorithm was run. We do not know ahead of time which unifiers we will have to create, so we encode everything that can possibly occur when the unification algorithm is run. To encode the absence of a cycle, we encode the existence of a path from one clause to another and the fact that there is no path from a clause to itself. This requires encoding all possible transitivity and irreflexivity axioms that may occur.

Our implementation, ChewTPTP-SMT, is a command line program written in C++ that allows the user to decide whether to encode the problem as a SAT problem or an SMT problem. If the user chooses SMT, our implementation uses Yices to test the satisfiability of the encoding. If the user chooses SAT, then the user can also choose whether to test the satisfiability using Yices or Minisat, with a DIMACS encoding of SAT.

Experimental Results

We tested our prover in all three settings on a subset of TPTP [79] problems. Tables 1-4 provide empirical data from these tests. SMT-Y denotes our prover run in SMT mode, SAT-Y is SAT mode using Yices, and SAT-M is SAT mode using Minisat. For Horn clauses, we ran ChewTPTP on all the Horn problems in the TPTP database, but for non-Horn we only had time to run it through the GRP problems. We report all problems that both provers solved within five minutes but SAT-M took greater than one second. We believe the problems in these tables are representative of the overall results. Columns in the table show the running time of each method, the clause generation time rounded off to the nearest second, the number of clauses generated, and the number of variables generated for each method. We also show whether or not the problem is rigidly satisfiable. For these experiments, we only tested rigid satisfiability with one instance of each clause.

We wanted to see if working modulo theories would improve the performance of

ChewTPTP. In the Horn case the running time was reduced significantly, except for a small percentage of exceptions. In the non-Horn case, working modulo theories often increased the running time. Generally, Yices was faster than Minisat on SAT problems without theories.

We believe we have an explanation for our results. In the Horn problems the number of clauses is reduced by an order of magnitude, whereas in the non-Horn problems the number of clauses is not reduced by much. This implies that working modulo theories is only useful when the clauses size is reduced significantly.

In the Horn encoding, everything can be encoded in $O(n^2)$ except for the encoding of unification and acyclicity, which require $O(n^3)$ space. When we remove the clauses used to represent unification and acyclicity, the number of clauses is now $O(n^2)$. However, for the encoding of non-Horn clauses, we must encode the fact of a leaf node having a complementary literal as an ancestor. This encoding is $O(n^3)$. We do not know how to encode this using the theories of Yices, so we have kept the propositional encoding. Therefore, when we remove the encoding of unification and acyclicity, the entire coding of the problem is still $O(n^3)$. We conjecture a good rule of thumb for deciding when it is useful to encode properties using theories. We conjecture that if the number of clause can be reduced by a factor of n, then the coding is useful, but if the asymptotic complexity remains the same, then it is not a good idea.

Implementation Status

The current version of ChewTPTP-SMT is a sound and complete first-order logic theorem prover. Though ChewTPTP-SMT runs well on Horn problems, it is not currently a competitive solution to the first-order validity problem compared to other theorem provers. Similar to ChewTPTP-SAT, eliminating restarts will provide a significant improvement to the performance ChewTPTP-SMT. Additional improvement in performance would be seen if the non-Horn encoding can be reduced by a factor of n, however at this point, we have

Table 3.2: ChewTPTP Times For Horn Problems

Name	SAT-M/Y Clause Gen	SMT-Y Clause Gen	SAT-M Total	SAT-Y Total	SMT-Y Total
PUZ008-1.p	1	0	1.06	0.89	0.11
NLP106-1.p	2	0	1.8	1.9	0.06
NLP104-1.p	2	0	1.82	1.9	0.05
NLP105-1.p	2	0	1.83	1.89	0.06
NLP107-1.p	2	0	2.47	1.99	0.06
GRP033-3.p	1	0	2.48	1.8	0.28
NLP109-1.p	1	0	2.49	1.99	0.05
NLP113-1.p	2	0	2.51	2.01	0.06
NLP110-1.p	2	0	2.74	1.84	0.07
NLP112-1.p	2	0	2.92	1.92	0.07
NLP111-1.p	1	0	2.94	1.93	0.06
NLP108-1.p	2	0	2.94	1.94	0.07
PUZ036-1.005.p	3	0	4.33	2.92	0.03
RNG037-2.p	4	0	5.33	5.35	6.2
RNG038-2.p	4	0	5.34	3.89	19.94
RNG001-5.p	4	0	6.93	5.32	0.84
SWV015-1.p	9	0	9.64	10.08	0.08
SWV017-1.p	11	0	10.82	11.27	0.1
RNG006-2.p	7	0	11.19	7.53	6.03

Table 3.3: ChewTPTP Clause and Variable Count For Horn Problems

Name	SAT-M/Y Cls Ct	SMT-Y Cls Ct	SAT-M/Y Var Ct	SMT-Y Var Ct	Result
PUZ008-1.p	52957	323	207608	216	sat
NLP106-1.p	130174	338	513774	392	unsat
NLP104-1.p	130724	344	515712	398	unsat
NLP105-1.p	130724	344	515712	398	unsat
NLP107-1.p	137380	315	542996	370	unsat
GRP033-3.p	115013	737	445065	383	sat
NLP109-1.p	137380	315	542996	370	unsat
NLP113-1.p	137897	319	544836	374	unsat
NLP110-1.p	128150	296	506951	350	unsat
NLP112-1.p	135667	287	537099	342	unsat
NLP111-1.p	135667	287	537099	342	unsat
NLP108-1.p	135667	287	537099	342	unsat
PUZ036-1.005.p	185292	45	729464	91	unsat
RNG037-2.p	221760	1524	876393	714	sat
RNG038-2.p	230063	1522	910786	718	sat
RNG001-5.p	258888	1527	1026821	725	sat
SWV015-1.p	559284	1047	2105121	532	unsat
SWV017-1.p	625119	1137	2354882	578	unsat
RNG006-2.p	432194	2058	1702459	925	sat

Table 3.4: ChewTPTP Times For Non-Horn Problems

Name	SAT-M/Y Clause Gen	SMT-Y Clause Gen	SAT-M Total	SAT-Y Total	SMT-Y Total
ANA025-2.p	1	0	1.02	1.04	2.43
COL121-2.p	0	1	1.02	0.92	1.41
ANA004-4.p	1	0	1.33	1.87	2.77
GRA001-1.p	2	2	1.92	1.74	4.08
ANA029-2.p	2	2	2.05	2.08	4.68
ANA005-2.p	2	1	2.38	2.31	4.72
ANA004-2.p	2	1	2.39	2.3	5.06
ANA003-2.p	3	1	2.96	2.81	5.53
GRP123-1.003.p	3	2	3.41	3.76	18.11
ANA001-1.p	4	2	4	3.84	7.94
GRP123-2.003.p	4	3	5.55	5.37	17.66
ANA002-2.p	5	3	5.73	5.34	10.56
ANA002-1.p	5	3	6.17	5.67	11.84
GRP124-2.004.p	9	6	10.51	11.4	43.91
GRP033-3.p	15	6	20.11	15.69	23.18
GRP123-3.003.p	28	20	30.63	30.73	80.84
ALG002-1.p	1	1	43.51	64.92	75.33
ANA004-5.p	2	1	47.25	21.5	83.54
GRP124-3.004.p	46	31	88.23	83.83	171
COM003-2.p	82	49	88.72	84.54	168.1

Table 3.5: ChewTPTP Clause and Variable Count For Non-Horn Problems

Name	SAT-M/Y Cls Ct	SMT-Y Cls Ct	SAT-M/Y Var Ct	SMT-Y Var Ct	Result
ANA025-2.p	41129	36020	2655	2286	sat
COL121-2.p	47725	20335	2322	1538	sat
ANA004-4.p	44142	36844	3160	2631	sat
GRA001-1.p	64222	60849	3292	3161	sat
ANA029-2.p	79860	66884	4107	3388	sat
ANA005-2.p	93806	68206	4907	3802	unsat
ANA004-2.p	93806	68206	4907	3802	unsat
ANA003-2.p	114945	78930	5654	4243	unsat
GRP123-1.003.p	111866	94335	4589	3596	unsat
ANA001-1.p	154246	113596	6680	5185	unsat
GRP123-2.003.p	180783	154243	6723	5450	unsat
ANA002-2.p	226149	151313	7457	5436	unsat
ANA002-1.p	229871	151313	7544	5437	unsat
GRP124-2.004.p	339070	283967	10854	8953	unsat
GRP033-3.p	699160	301901	15989	8961	sat
GRP123-3.003.p	1003831	934044	17763	15377	unsat
ALG002-1.p	54559	32731	3524	2460	unsat
ANA004-5.p	101166	44953	4981	3196	unsat
GRP124-3.004.p	1596801	1468732	25314	21981	unsat
COM003-2.p	2920669	2365922	46818	36051	sat

yet to develop such an encoding.

3.4 Conclusion

We introduced in [126] and [128] two novel approaches to first-order theorem proving which encodes a closed rigid connection tableaux proof of first-order unsatisfiability in SAT and SMT respectively.

Compared to our encoding in SAT, the encoding in SMT is more natural and more efficient. As part of our encoding, we need to encode the solving of unification problems and the acyclicity of the tableau. In SAT, it was necessary to add cubically many clauses to encode the solving of unification. In addition, it was necessary to add cubically many clauses to encode the acyclicity of the tableau. However, when encoding this information in SMT, there was no need to encode the solving of unification, since this was accomplished directly with the Yices recursive datatype theory. The number of unification clauses was reduced from a cubic to a quadratic number. Similarly for acyclicity of tableau, we did not need to encode the transitivity and irreflexivity of the path relation. We only needed to express edges in the tableau as inequalities. The number of clauses to represent acyclicity also dropped from a cubic number to a quadratic number.

In the SMT Horn encoding, all the other information in the tableau can also be encoded with a quadratic number of clauses. Therefore the entire encoding of the existence of a tableau dropped from a cubic number of clauses in SAT to a quadratic number in SMT. This drastically reduced the number of clauses, and simultaneously decreased the time needed to decide the satisfiability of the clauses. There was only a small reduction in number of clauses for non-Horn clauses, because we still need to encode the fact that all paths in the tableau can be closed. Therefore the entire encoding is still cubic, and the running time was actually worse. We conjecture a rule of thumb saying that it is worthwhile to use theories if the number of clauses is reduced by a factor of n, but not worthwhile if the asymptotic

number remains the same.

For future work includes looking at ways to be able to use SMT to further reduce the representation for non-Horn clauses, ideally cutting it down to a quadratic number of clauses. It would be possible to define a theory to do this directly, but we have not yet figured out how to do it with the existing theories in Yices. In addition, in order to prove the general first order problem we also need to find a good heuristic to decide exactly which clauses should be copied. We would like a method to decide satisfiability from rigid satisfiability. It would be useful to have an encoding of rigid clauses modulo a non-rigid theory, as discussed in [122]. This way, we could immediately identify some clauses as non-rigid, and work modulo those clauses.

Though our implementations are not yet state-of-the-art, our work shows the usefulness of SAT and SMT to theorem proving in first order logic. We suspect there are other logics which could also be solved efficiently using these methods.

Chapter 4

Combining Instance Generation and Resolution

Although resolution methods appear to be more efficient in practice, there are some classes of problems that are suited better for instantiation-based methods. In [136] we show that we can combine both instance generation and resolution into a single inference system while retaining completeness with the aim of getting the best of both methods. We define the inference system named $\mathrm{SIG} - \mathrm{Res}$ that combines semantic selection instance generation (SInst-Gen) with ordered resolution.

Each clause in the given set of clauses is determined, by some heuristic, to be an instantiation clause and placed in the set P or a resolution clause and placed in the set R or placed in both P and R. Clauses from P are given to a SAT solver and inferences among them are treated as in SInst-Gen, while any inference which involves a clause in R is a resolution inference.

Our combination of instance generation and resolution differs from the method used in the instantiation-based theorem prover iProver [133] which uses resolution inferences to simplify clauses, i.e. if a conclusion of a resolution inference strictly subsumes one if its premise then the conclusion is added to the set of clauses sent to the SAT solver and the

subsumed premise is removed. Our inference system also allows for the use of resolution for the simplification of the clauses in P, but differs from iProver in that it restricts certain clauses, the clauses in R, from any instance generation inference.

Our idea is similar to the idea of Satisfiability Modulo Theories (SMT), where clauses in P represent data, and the clauses in R represent a theory. This is similar to the SMELS inference system [132] and the DPLL($\Gamma + T$) inference system [141]. The difference between those inference systems and ours is that in those inference systems, P must only contain ground clauses, and the theory is all the nonground clauses, whereas in our case we allow nonground clauses in P.

Below, we discuss SIG-Res, prove the completeness of our inference system, discuss our implementation called Spectrum, and present some initial results.

4.1 Preliminaries

4.1.1 Jeroslow Constant

$\perp$ is used to denote a distinguished constant called the *Jeroslow constant* and the substitution which maps all variables to $\perp$. If L is a literal then $L\perp$ denotes the ground literal obtained by applying the $\perp$-substitution to L and if P is a set of clauses then $P\perp$ denotes the set of ground clauses obtained by applying the $\perp$-substitution to the clauses in P.

4.1.2 Term Orderings

A *binary relation* on a set A is a set of ordered pairs of elements from A. A binary relation $\succ$ is *transitive* if for all x, y and z in A it holds that if $x \succ y$ and $y \succ z$ then $x \succ z$. A binary relation $\succ$ is *irreflexive* if for all x in A it holds that **not** $x \succ x$.

Ordering are binary relations with special properties. A *strict partial ordering* is an irreflexive and transitive binary relation. A strict ordering is *well-founded* if there is no

infinite descending chain of elements. An ordering $\succ$ is *total* on S if for every distinct pair of elements x and y in S it holds that $x \succ y$ or $y \succ x$. An ordering $\succ$ is *stable under substitution* if for any substitution σ and for all x and y in S it holds that if $x \succ y$ then $x\sigma \succ y\sigma$.

Given a signature Σ we say that $\succ$ is *compatible with Σ-operations* if $s \succ s'$ implies $f(t_1, \cdot, t_{i-1}, s, t_{i+1}, \cdot, t_n) \succ f(t_1, \cdot, t_{i-1}, s', t_{i+1}, \cdot, t_n)$ for all $f/n \in \Sigma$, for all terms $s, s', t_1, \cdot, t_n \in T_{\Sigma \cup X}$ and for all coefficients $i \in \mathbb{N}, 1 \leq i \leq n$.

We say that $\succ$ has the *subterm property* if $s \succ s'$ whenever s' is a proper subterm of s. We say $\succ$ is a *rewrite relation* if $\succ$ is compatible with Σ-operations and stable under substitutions and we say $\succ$ is a *rewrite ordering* if it is a strict partial ordering and a rewrite relation. A *simplification ordering* is a rewrite ordering that has the subterm property.

An ordering $>$ on terms is any strict partial ordering that is well-founded, stable under substitution and total on ground terms. We extend $>$ to atoms in such a way so that for any atom A we have $\neg A > A$. The ordering $>$ is extended to clauses by considering a clause as a multiset of literals.

Given a clause C, a literal $L \in C$ is *maximal* in C if there is no $K \in C$ such that $K > L$. We define a mapping, $\max$ from clauses to multisets of literals such that $\max(C) = \{L | L \text{ is maximal in } C\}$.

If X is a set of variables, Σ is a signature, and $>$ is a strict partial ordering on Σ a *(regular) symbol weight assignment* is a function $\lambda : \Sigma \cup X \rightarrow \mathbb{N}$. Furthermore, we say λ is *admissible* for $>$ if and only if

(i) $\exists \lambda_0 \in \mathbb{N}^{>0}$ such that $\forall x \in X : \lambda(x) = \lambda_0$ and $\forall c/0 \in \Sigma : \lambda(c) \geq \lambda_0$

(ii) If there is $f/1 \in \Sigma$ such that $\lambda(f) = 0$, then we must have: $\forall g \in \Sigma : f \geq g$

Throughout the remainder of this paper and in our implementations we use the simplification ordering on terms called the Knuth-Bendix ordering. It is parameterized by an ordering on the symbols in the signature and a weight function on terms. We give below the version found in [127].

We extend a weight assignment λ to a function $w_\lambda : T_{\Sigma \cup X} \to \mathbb{N}$ on terms as follows:

- For $x \in X$:

$$w_\lambda(x) = \lambda(x)$$

- For $n \in \mathbb{N}$ and $t_1, \cdots, t_n \in T_{\Sigma \cup X}$:

$$w_\lambda(f(t_1, \cdots, t_n)) = \lambda(f) + \sum_{i=1}^{n} w_\lambda(t_i)$$

Definition 4 *(Knuth-Bendix Ordering) Let Σ be a signature and let X be a set of variables. Additionally, let $>$ be a strict partial ordering, the precedence, on Σ and $\lambda : \Sigma \cup X \to \mathbb{N}$ be a regular symbol weight assignment that is admissible for $>$. Finally, let $w = w_\lambda : T_{\Sigma \cup X} \to \mathbb{N}$ be the regular term weight function induced by λ.*

We define the Knuth-Bendix ordering $\succ_{KBO} \subseteq T_{\Sigma \cup X} \times T_{\Sigma \cup X}$ induced by $(>, \lambda)$ on terms $s, t \in T_{\Sigma \cup X}$ in the following way: $s \succ_{KBO} t$ if and only if

(KBO1) $\forall x \in X : |s|_x \geq |t|_x$ *and* $w(s) > w(t)$

or

(KBO2) $\forall x \in X : |s|_x \geq |t|_x$, $w(s) > w(t)$ *and one of the following cases holds:*

- **(KBO2a)** $\exists f/1 \in \Sigma, \exists x \in X, \exists n \in \mathbb{N}^{>0}$ *such that* $s = f^n(x)$ *and* $t = x$.
- **(KBO2b)** $\exists f/m, g/n \in \Sigma (m, n \in \mathbb{N}), \exists s_1, \cdots, s_m, t_1, \cdots, t_n \in T_{\Sigma \cup X}$ *such that* $s = f(s_1, \cdots, s_m), t = g(t_1, \cdots, t_n)$ *with* $f > g$
- **(KBO2c)** $\exists f/m \in \Sigma (m \in \mathbb{N}^{>0}), \exists s_1, \cdots, s_m, t-1, \cdots, t_m \in T_{\Sigma \cup X}, \exists i, 1 \leq i \leq m$ *such that* $s = f(s_1, \cdots, s_m), t = f(t_1, \cdots, t_m)$ *and such that* $s = t_1, \cdots, s_{i-1} = t_{i-1}, s_i \succ_{KBO} t_i$

4.1.3 Interpretations

A *Herbrand interpretation*, I, is a consistent set of ground literals. We say that a ground literal is *undefined* in I if neither it nor its complement is in I. If a ground literal L is in I then we say that L is *true* in I and $\overline{L}$ is *false* in I. I is a *total interpretation* if no ground literal is undefined in I. A ground clause C is true in a partial interpretation I if there exists some literal L in C that is true in I, and we say that C is satisfied by I.

4.1.4 Closures

A *closure* is denoted by the pair $C' \cdot \sigma$, where C' is a clause and σ is a substitution. Suppose $C = C' \cdot \sigma$ is a closure. As an abuse of notation we may also refer to C as the instance of C' under the substitution σ, that is $C'\sigma$. We say that C is a *ground closure* if $C'\sigma$ is ground. If S is a set of clauses and $C' \in S$ we say that $C'\sigma$ is an *instance of* S. A *closure ordering* is any well founded and total (modulo renaming) ordering on closures.

4.2 Semantic Selection Instance Generation and Ordered Resolution

The main idea behind all saturation-based instance generation methods is to augment a set of clauses with sufficiently many proper instances so that the satisfiability of the set can be determined by a SAT solver. Additional instances are generated using some form of the Inst-Gen [106] inference rule. An instance generation with semantic selection inference system (SInst-Gen) (See Figure 4.1) uses a selection function and the notion of *conflicts* to determine exactly which clauses are to be used as premises in the instance generation inferences.

Let P be a set of first order clauses and view $P\bot$ as a set of propositional clauses. Under this setting, if $P\bot$ is unsatisfiable, then P is unsatisfiable and our work is done.

Otherwise a model for $P\bot$ is denoted as $I_\bot$ and we define a selection function, $\mathrm{sel}(C, I_\bot)$, which maps each clause $C \in P$ to a singleton set $\{L\}$ such that $L \in C$ and $L\bot$ is true in $I_\bot$.

We say, given a model $I_\bot$, that two clauses $L \vee \Gamma$ and $\overline{K} \vee \Delta$ *conflict* if

(i) $L \in sel(L \vee \Gamma, I_\bot)$ and $\overline{K} \in \mathrm{sel}(\overline{K} \vee \Delta, I_\bot)$

(ii) L and K are unifiable

Instance generation with semantic selection methods saturate a set of clauses P by repeatedly calling upon a SAT solver to obtain a model for $P\bot$ and resolving all conflicts with SInst-Gen inferences. If $P\bot$ is ever found unsatisfiable, P is unsatisfiable. If, on the other hand, $P\bot$ is satisfiable and no conflicts exist then P is satisfiable. SInst-Gen is refutationally complete [106] but may not halt.

$$\frac{L \vee \Gamma \qquad \overline{K} \vee \Delta}{(L \vee \Gamma)\sigma \qquad (\overline{K} \vee \Delta)\sigma} \text{ (SInst-Gen)}$$

where

1. $L \in sel(L \vee \Gamma, I_\bot)$ and $\overline{K} \in \mathrm{sel}(\overline{K} \vee \Delta, I_\bot)$
2. $\sigma = \mathrm{mgu}(L, K)$

Figure 4.1: SInst-Gen Inference Rule

The ordered resolution and factoring inference rules are well known in the literature. For completeness they are given in Figure 4.2. The strength of ordered resolution is in its ability to reduce the search space by requiring only inferences between clauses which conflict where $\max$ is the selection function.

The satisfiability of a set R is determined by applying resolution and factoring inferences rules to the clauses in R in a fair manner until either the empty clause ($\bot$) is resolved, in which case R is unsatisfiable, or the set is saturated and $\bot \notin R$, in which case R is satisfiable. As is the case with SInst-Gen, ordered resolution with factoring is refutationally complete, but for some satisfiable problems may not halt.

$$\frac{L \vee \Gamma \qquad \overline{K} \vee \Delta}{(\Gamma \vee \Delta)\sigma} \text{ (Ordered Resolution)}$$

where

1. $L \in \max(L \vee \Gamma)$ and $\overline{K} \in \max(\overline{K} \vee \Delta)$
2. $\sigma = \mathrm{mgu}(L, K)$

$$\frac{L \vee K \vee \Delta}{(L \vee \Delta)\sigma} \text{ (Factoring)}$$

where $\sigma = \mathrm{mgu}(L, K)$

Figure 4.2: Ordered Resolution and Factoring Inference Rules

4.3 SIG-Res

The inferences in SIG-Res are variations of SInst-Gen, ordered resolution and factoring (see Figure 4.3). SIG-Res is an inference system that establishes two sets of clauses. Given a problem in CNF, S, which we wish to prove satisfiable or unsatisfiable, we create two sets of clauses, $P \subseteq S$ and $R \subseteq S$, not necessarily disjoint, such that $P \cup R = S$. Given some clause $C \in S$, C is designated as either a clause in P, a clause in R, or both, according to any distribution heuristic of our choosing, so long as $P \cup R = S$.

The distribution heuristic is a key mechanism in this inference system as it determines which inferences are applied to the clauses. Under SIG-Res, a distribution heuristic can, at one end of the spectrum, insert all the clauses of S in P, leaving R empty, which would make the system essentially a instance generation inference system. On the other end of the spectrum, the distribution heuristic can distribute all the clauses to R, leaving P empty, making the system a resolution system. This flexibility allows any number of heuristics to be used and heuristics to be tailored to specific classes of problems. An open question is which heuristics perform best and for which classes of problems. In Section 4.5 we describe one general heuristic, GSM, which we have incorporated into our implementation.

The selection function, $\mathrm{sel}(C, I_\perp)$, where $C \in P \cup R$ and $I_\perp$ is a model for $P\perp$, is defined as follows. For clarity, we note that $\mathrm{sel}(C, I_\perp)$ returns a singleton set if $C \in P$ and

a non-empty set if $C \in R$.

$$\mathrm{sel}(C, I_\perp) = \begin{cases} \{L\} \text{ for some } L \in C \text{ such that } L\perp \in I_\perp & \text{if } C \in P \\ \max(C) & \text{if } C \in R \end{cases}$$

We will have the usual redundancy notions for saturation inference systems. We can define deletion rules to say that a clause can be deleted if it is implied by zero or more smaller clauses. For example, tautologies can be deleted. The clause ordering, as we will define it in the next section, will restrict what subsumptions can be done. In particular, if a clause C is in R, we say that C is subsumed by a clause D if there exists a substitution σ such that $D\sigma$ is a subset of C. If C is a clause in P, we say that C is subsumed by D if there exists a substitution σ such that $D\sigma$ is a proper subset of C.

We will define saturation in the next section, to take into account the model $I_\perp$. Saturation of S under SIG-Res is achieved by ensuring that all possible inferences are made (fairness). One way to ensure fairness, as is done in the Primal Partial Instantiation method, is to increment a counter and only allow inferences with premises having depth less than or equal to the counter. An alternative method is to perform all possible inferences with the exception that we restrict conclusions generated during each iteration from being considered as premises until the next iteration. We have implemented IG-Res in a theorem prover called Spectrum. Our implementation uses the latter method and follows Algorithm 5.

4.4 Completeness

Let S be a set of clauses. We begin by defining an ordering $\prec$ on the closures in S. Given an ordering on terms, $<$, we denote by $\prec_C$ any closure ordering with the following properties: for any closures $C \cdot \sigma$ and $D \cdot \tau$, $C \cdot \sigma \prec_C D \cdot \tau$ if

i. $C\sigma < D\tau$ or

ii. $C\sigma = D\tau$ and $C = D\rho$ where ρ is a proper instantiator of D

$$\frac{L \vee \Gamma \qquad \overline{K} \vee \Delta}{(L \vee \Gamma)\sigma \qquad (\overline{K} \vee \Delta)\sigma} \text{ (SInst-Gen)}$$

where

1. $L \vee \Gamma \in P$ and $\overline{K} \vee \Delta \in P$
2. $L \in \mathrm{sel}(L \vee \Gamma, I_\perp)$ and $\overline{K} \in \mathrm{sel}(\overline{K} \vee \Delta, I_\perp)$
3. $\sigma = \mathrm{mgu}(L, K)$
4. $(L \vee \Gamma)\sigma \in P$ and $(\overline{K} \vee \Delta)\sigma \in P$

$$\frac{L \vee \Gamma \qquad \overline{K} \vee \Delta}{(\Gamma \vee \Delta)\sigma} \text{ (Ordered Resolution)}$$

where

1. $L \vee \Gamma \in R$ or $\overline{K} \vee \Delta \in R$
2. $L \in \mathrm{sel}(L \vee \Gamma, I_\perp)$ and $\overline{K} \in \mathrm{sel}(\overline{K} \vee \Delta, I_\perp)$
3. $\sigma = \mathrm{mgu}(L, K)$
4. $(\Gamma \vee \Delta)\sigma \in P$ if $L \vee \Gamma \notin R$ or $\overline{K} \vee \Delta \notin R$

$$\frac{L \vee K \vee \Delta}{(L \vee \Delta)\sigma} \text{ (Factoring)}$$

where

1. $\sigma = \mathrm{mgu}(L, K)$
2. $(L \vee \Delta)\sigma \in P$ if $L \vee K \vee \Delta \notin R$

Figure 4.3: SIG-Res Inference Rules

We denote by $\prec_S$ any (subsumption) closure ordering with the following property: for any closures $C \cdot \sigma$ and $D \cdot \tau$, $C \cdot \sigma \prec_S D \cdot \tau$ if

i. $C\sigma < D\tau$ or

ii. $C\sigma = D\tau$ and $C\rho = D$ where ρ is a proper instantiator of C

Given a set of clauses $S = P \cup R$ and orderings $\prec_C$ and $\prec_S$ we define the ordering $\prec$ on the closures of S as follows. For all closures C and D of S, $C \prec D$ iff

i. C and D are closures of P and $C \prec_C D$ or

ii. C and D are closures of R and $C \prec_S D$ or

iii. C is a closure of P and D is a closure of R

$C \cdot \sigma$ is a *minimal closure* in S if C is a closure in S and C is the minimal representation of $C'\sigma$ in S under $\prec$.

A ground clause C is *redundant* in S if there are clauses $C_1, \cdots, C_n$ in set $Gr(S)$ such that $C_i \prec C$ holds for all i, $1 \leq i \leq n$ and $C_1, \cdots, C_n \vDash C$. A clause C is redundant in S if all of the ground instances of C are redundant in $Gr(S)$. A *derivation* of an inference system is a sequence $(S_0, I_0, \mathrm{sel}_0), \cdots, (S_i, I_i, \mathrm{sel}_i), \cdots$, where each S_i is a multiset of clauses divided into sets P_i and R_i, I_i is a model of $P_i\bot$, sel_i is a selection function based on the model I_i, and S_{i+1} results from applying an inference rule or deletion rule on S_i. The sequence has as its limit the set of *persistent clauses* $S_\infty = \bigcup_{i\geq 0} \bigcap_{j \geq i} S_j$. By definition of redundancy, if a clause is redundant in some S_i it is redundant in S_∞.

We define a *persistent model* I_∞ in the following way. Let $A_1, A_2, \cdots$ be an enumeration of all the atoms. Let D_0 be the derivation sequence. For each i, let D_i be the subsequence of D_{i-1} such that (i) if A is true in an infinite number of I_j then D_i is the subsequence of D_{i-1} that only contains tuples $(S_j, I_j, \mathrm{sel}_j)$ where I_j makes A true, else (ii) if A is not true in an infinite number of S_j then D_i is the subsequence of D_{i-1} that only contains tuples $(S_j, I_j, \mathrm{sel}_j)$ where I_j makes A false. If $D_\infty = (S_0, I_0, \mathrm{sel}_0), \cdots, (S_i, I_i, \mathrm{sel}_i), \cdots$, then we define $I_\infty = \bigcup_{j\geq 0} I_j$.

S_∞ is called *saturated* if the conclusion of every inference of $(S_\infty, I_\infty, \mathrm{sel}_\infty)$ is in S_∞ or is redundant in S_∞. A derivation is *fair* if no inference is ignored forever, i.e. the conclusion of every inference among persistent clauses is persistent or redundant in S_∞. A fair derivation produces a saturated set.

Now we define the construction of a candidate model for the ground instances of S. Given a clause ordering $\prec$ on the closures of a set of clauses, $S = P \cup R$, for every ground closure, C, of S, we define ϵ_C as a set of zero or more literals in C. We say that C is *productive* if $\epsilon_C \neq \varnothing$, otherwise we say that C is not productive.

Let D be a ground closure in S. We define $I_D = \bigcup_{C \prec D} \epsilon_C$ where C is a ground closure of S and define $I^D = I_D \cup \epsilon_D$. It follows that if $C \prec D$ then $I_C \subseteq I_D$. We define $I_S = \bigcup_C \epsilon_C$ where C is a ground closure in S.

Suppose that $P\bot$ is satisfied by the model $I_\bot$ and let $\prec$ be a closure ordering on the closures of S. We construct a candidate model for the ground instances of S as follows. For all ground closures $C = C' \cdot \sigma$ we define $\epsilon_C = \{L\sigma\}$ if

i. $C'\sigma$ is not true in I_C and

ii. $L\sigma$ is undefined in I_C and

iii. ($C' \in P$ and $L\bot \in I_\bot$) or ($C' \in R$ and $\max(C'\sigma) = \{L\sigma\}$)

Otherwise $\epsilon_C = \varnothing$.

Theorem 6 *Let $S = P \cup R$ be a multiset of clauses saturated under SIG-Res. If $P\bot$ is satisfied by $I_\bot$ then the set of ground instances of P is satisfiable in the candidate model I_S.*

Proof Let $S = P \cup R$ be a multiset of clauses saturated under SIG-Res and suppose $P\bot$ is satisfied by $I_\bot$. By the completeness of SInst-Gen [106], I_P is a model of the ground instances of P. As $I_P \subseteq I_S$ and I_S is consistent, it follows that the set of ground instances of P is satisfiable in the candidate model I_S.

Theorem 7 *Let $S = P \cup R$ be a multiset of clauses saturated under SIG-Res. S is satisfiable if $P\bot$ is satisfied by $I_\bot$ and S does not contain the empty clause.*

Proof Let $S = P \cup R$ be a multiset of clauses saturated by SIG-Res. Suppose $P\bot$ is satisfied by $I\bot$ and S does not contain the empty clause. We claim that I_S is a model of all ground instances of S.

Suppose on the contrary that I_S is not a model for the set of ground instances of S. Let $C = C' \cdot \sigma$ be the minimal ground closure of S that is false or undefined in I_S.

As $P\bot$ is satisfied by $I_\bot$, it follows that the set of ground instances of P is satisfiable in the candidate model I_S. Therefore it must be the case that $C' \in R$.

Let $L\sigma \in \max(C)$. Now, suppose $L\sigma$ is undefined in I_S. Then as $C'\sigma$ is not true in I_S, C is productive, a contradiction. Hence, $L\sigma$ is false in I_S. If $L\sigma$ is a duplicate in $C'\sigma$ then let B' be the conclusion resulting from the factoring of C'. Then $B'\sigma$ is smaller than $C'\sigma$, thus contradicting the minimality of $C'\sigma$. Therefore, let us assume that $L\sigma$ is not a duplicate and let $C' = C'' \vee L$ for some C''.

As C is not productive and $L\sigma$ is false in I_S there exists some productive minimal ground closure $D = D' \cdot \sigma$ [1] such that $D \prec C$ and $\epsilon_D = \{\overline{L\sigma}\}$. Therefore $D' = D'' \vee K$ where $K\sigma = \overline{L\sigma}$ and $D''\sigma$ is not true in I^D.

Let $B' = (D'' \vee C'')\tau$ where $\tau = \mathrm{mgu}(\overline{K}, L)$ be the conclusion of the resolution inference with premises $D' = D'' \vee K$ and $C' = C'' \vee L$ and let B be the minimal representative of $B'\sigma$. Therefore B is a ground instance of B'.

Now $D' \in P$ or $D' \in R$ so we proceed by cases.

Case 1: Suppose that $D' \in P$. Since S is saturated by SInst-Gen and $C' \in R$, then the conclusion of the resolution inference between D' and C', i.e. B', is in P or is redundant.

If B' is in P then $B'\sigma$ is satisfied in I_S. If B' is redundant then there exists $B_1, B_2, ..., B_n \in P$ such that $B_1, B_2, \cdots, B_n \models B'$ and for all i, $1 \leq i \leq n$, B_i is smaller than B'. Since for all i, $1 \leq i \leq n$, $B_i\sigma$ is satisfied in I_S then $B'\sigma$ is satisfied in I_S.

[1] As clauses are standardized apart we use a single substitution σ.

Since $B'\sigma = (D'' \vee C'')\sigma$ is true in I_S and $C''\sigma$ is not true in I_S then $D''\sigma$ must be satisfied in I_S. Now as $D''\sigma$ is not true in I^D, then it follows that $D \prec B$. Therefore $(D'' \vee K)\sigma$ is smaller than $(D'' \vee C'')\sigma$. Hence $K\sigma = \overline{L\sigma}$ is smaller than $C''\sigma$, which is a contradiction as $L\sigma \in \max(C)$.

Case 2: Suppose now that $D' \in R$. Since $\epsilon_D = \{K\sigma\}$, $K\sigma \in \max(D)$. Therefore $D''\sigma$ is smaller than $K\sigma$. Hence B' is strictly smaller than C. And as $D''\sigma$ is not true in I_S and $C''\sigma$ is not true in I_S we have $B'\sigma$ is not true in I_S. If B' is in S, this contradicts the minimality of C.

If B' is redundant in S then there exists clauses $B_1, B_2, \cdots, B_n \in S$ each smaller than B' such that $B_1, B_2, \cdots, B_n \models B'$. It follows that there exists some $0 \leq i \leq n$ such that $B_i\sigma$ is false in I_S, hence a contradiction. ∎

Since SIG-Res is refutationally complete and the inferences are sound, it should be clear that it is only necessary that *at some point in time* we insert the conclusions of SIG-Res inferences into the appropriate set as defined by the inference rules. Prior to that time, without affecting completeness, we can insert conclusions from inferences into P or R with disregard to the algorithm if by doing so we can find a solution quicker.

4.5 Spectrum

We have implemented SIG-Res in a theorem prover for first order logic called Spectrum. The name comes from the fact that given a set of clauses, our choices to construct the sets P and R are among a spectrum.

Spectrum is written in C++, has a built-in parser for CNF problems in the TPTP format [79] and outputs results in accordance to the SZS ontology [79]. It takes as arguments a filename and mode and outputs satisfiable or unsatisfiable. The modes determine how the clauses will be distributed to the sets P and R. There are a number of distribution modes which Spectrum can be run in. When running Spectrum with the `-p` flag, Spectrum

places all clauses in P, hence makes Spectrum run essentially as an instantiation-based theorem prover. The flag `-r` makes Spectrum run essentially as a resolution theorem prover by placing all the clauses in R. Running spectrum without a mode flag runs our default heuristic we call Ground-Single Max (GSM).

It is well known that in general, SAT solvers are more efficient in solving ground instances than resolution. Our GSM heuristic takes advantage of this by placing all ground clauses in P. GSM also places all clauses with more than one maximal literal in P. GSM places all other clauses in R.

When the program begins, the program distributes the clauses to the two sets P and R in accordance with the distribution mode and if a clause is inserted in R, its maximal literals are identified. After distributing the clauses, Spectrum follows Algorithm 5.

As we begin the instance generation phase on the set P, Yices [147] is used to check the satisfiability of the ground instances of $P\bot$. If Yices reports the problem as inconsistent, Spectrum reports unsatisfiable and halts. However, if Yices reports the problem is consistent (satisfiable) we retrieve a model from Yices and select for each clause the first literal in the clause whose propositional abstraction is true in the model. These are the selected literals that we use for determining if conflicts exist. If a conflict exists we instantiate the new clauses and check to see if the new clauses already exist in P. If not, we add them to P. To ensure that we do not run the instance generation phase forever we do not allow conclusions to SInst-Gen inferences to be premises until after the next call to the SAT solver.

Following the instance generation phase, we check for resolution inferences. We first resolve all unchecked pairs of clauses where both clauses are in R, and then for the unchecked pairs where one clause is in P and the other is in R. To ensure fairness, we exclude from being premises SInst-Gen conclusions that were added during the previous instantiation phase and conclusions from resolution and factoring inferences that are added in the current iteration. If an inference is made, we check to see if it is the empty clause.

If so, Spectrum reports unsatisfiable and halts. Otherwise, if one of the premises is in P we perform the simple redundancy check as stated above and when appropriate add the conclusion to P. If, on the other hand, both premises are in R we check for factors. If a factor is slated for R we determine if it already exists in R and if it is forwardly-subsumed by some clause in R. If it is slated for P we only check to see if it already exists in P.

If no new clause is added during an iteration, Spectrum reports Satisfiable and halts, otherwise it repeats the process.

Experimental Results

We have tested Spectrum on 450 unsatisfiable problems rated $easy$ in the TPTP library. These problems, in general, are not challenging for *state of the art* theorem provers, but allow us to compare the different modes of our implementation and give us simple proofs to analyze. Of the 450 problems we tested, Spectrum run in GSM mode for 300 seconds solved 192 problems [2]. Of these 192 problems when given the same time limit, 18 could not be solved by Spectrum run in `-p` mode where the problem is solved using only instance generation or in `-r` mode where only resolution inferences are allowed. Interestingly, 16 of these are in the LCL class of problems, the class of Propositional Logic Calculi. Many of these problems contain the axioms of propositional logic which have clauses that are similar to the transitivity property. These can produce a large number of clauses under resolution. These clauses, when run under our heuristic, are put in P to avoid this condition. Also present are clauses which we call *growing clauses* because their tendency to produce larger and larger clauses. These growing clauses, e.g. $\neg P(x) \vee P(f(x))$, contain pairs of complementary literals where each argument in the first is a subterm of the second and there exists at least one argument that is a proper subterm. Growing clauses, under our heuristic, since they have only a single maximal literal, are put in R which avoids this problem.

[2] These results reflect that our implementation is not yet competitive and lacks some key processes such as robust redundancy deletion.

Algorithm 5: Spectrum(P,R)

input : Sets P and R containing FO formula in conjunctive normal form
output: SATISFIABLE or UNSATISFIABLE
while $true$ **do**
    $N_P := \varnothing$;
    $N_R := \varnothing$;
    Run SAT on $P\bot$;
    if $P\bot$ *is unsatisfiable* **then return** UNSATISFIABLE;
    for $C_1, C_2 \in P$ **do**
        if $\text{conflict}(C_1, C_2) = true$ **then**
            $N_P := N_P \cup (\text{SInst-Gen}(C_1, C_2) \setminus P)$;
    for $C_1 \in P, C_2 \in R$ **do**
        $D := \text{Resolution}(C_1, C_2)$;
        if $\bot \in D$ **then**
            return UNSATISFIABLE;
        else if $D \neq \varnothing$ **then**
            $N_P := N_P \cup (D \setminus P)$;
    for $C_1, C_2 \in R$ **do**
        $D := \text{Resolution}(C_1, C_2)$;
        if $\bot \in D$ **then**
            return UNSATISFIABLE;
        else if $D \neq \varnothing$ **then**
            for $C \in D$ **do**
                $F := \text{Factor}(C)$;
                for $B \in F$ **do**
                    $T := \text{distribute}(B)$;
                    $N_T := N_T \cup (\{B\} \setminus T)$;
    if $N_P = \varnothing$ *and* $N_R = \varnothing$ **then**
        return SATISFIABLE;
    else
        $P := P \cup N_P$;
        $R := R \cup N_R$;

There are several examples from LCL, and also the GRP problem we illustrate below, where our heuristic performs better than solely using instance generation or resolution. The GRP problem is an example that contains clauses that can cause infinite growth, so it is not good for systems implementing only instance generation. While at the same time, it contains clauses similar to Transitivity where ordered resolution is explosive. We believe these examples show the use of our technique and the potential for further research into this area.

Example Proof

One problem in the TPTP library that illustrates another benefit of SIG-Res with the GSM heuristic is problem GRP006-1. Spectrum using our heuristic solved this problem in less than 1 second, but did not find a solution using instantiation or resolution alone. The initial distribution of clauses and an SIG-Res proof are given in Figure 4.4. As can be seen, by placing the clauses with more than one maximal literal, specifically clauses *3* and *4*, in P we avoid many resolution inferences that are not necessary for the proof. We also avoid generating many SInst-Gen inferences by placing clause 4 in P and clause 6 in R.

Before determining the problem unsatisfiable, Spectrum makes 3 passes through the while loop generating 32 clauses. During the initial iteration, no conflicts are found and resolution and factoring inferences produce a total of 9 new clauses. During the second iteration, 2 conflicts produce 2 new clauses and resolution and factoring produce 21 new clauses. During the third iteration, Yices returns back unsatisfiable as clause 2, 13 and 14 are inconsistent. This example shows that the clauses in a problem may have different properties and that by controlling the types of inferences that are applied to the clauses we may eliminates unnecessary inferences and may produce a solution sooner than if using resolution or instantiation inferences alone.

Clauses in P

1. $\neg E(inv(a))$

2. $E(a)$

3. $\neg P(x,y,z) \vee \neg P(y,w,v) \vee \neg P(x,v,t) \vee P(z,w,t)$

4. $\neg P(x,y,z) \vee \neg P(y,w,v) \vee \neg P(z,w,t) \vee P(x,v,t)$

Clauses in R

5. $\neg E(x) \vee \neg E(y) \vee \neg P(x,inv(y),z) \vee E(z)$

6. $P(inv(x),x,id)$

7. $P(x,inv(x),id)$

8. $P(x,id,x)$

9. $P(id,x,x)$

$$\frac{\neg E(x) \vee \neg E(y) \vee \neg P(x,inv(y),z) \vee E(z) \qquad P(x,inv(x),id)}{\textbf{10.}\ \neg E(x) \vee \neg E(x) \vee E(id)}\ \text{(Res(5,7))}$$

$$\frac{\neg E(x) \vee \neg E(x) \vee E(id)}{\textbf{11.}\ \neg E(x) \vee E(id)}\ \text{(Factor(10))}$$

$$\frac{\neg E(x) \vee \neg E(y) \vee \neg P(x,inv(y),z) \vee E(z) \qquad P(id,x,x)}{\textbf{12.}\ \neg E(id) \vee \neg E(x) \vee E(inv(x))}\ \text{(Res(5,9))}$$

$$\frac{E(a) \qquad \neg E(x) \vee E(id)}{\textbf{13.}\ \neg E(a) \vee E(id)}\ \text{(SInst-Gen(2,11))}$$

$$\frac{\neg E(inv(a)) \qquad \neg E(id) \vee \neg E(x) \vee E(inv(x))}{\textbf{14.}\ \neg E(id) \vee \neg E(a)}\ \text{(Res(1,12))}$$

Figure 4.4: Proof of GRP006-1

Implementation Status

Spectrum is a sound and complete first-order theorem prover but is not competitive compared to state-of-the-art theorem provers. Much of the disparity is due to the lack of maturity of Spectrum. Many theorem provers that rank high in competition, for example Vampire [102] and E [101], are more than a decade old thus have been fine tuned over the years. One major deficiency of Spectrum is the lack of redundancy elimination. Currently Spectrum only removes tautologies and performs forward subsumption checking on conclusions to resolution inferences. In order for Spectrum to compete, a comprehensive redundancy elimination routine must be established.

4.6 Conclusion

SIG-Res is a sound and complete inference system that combines $SInstGen$ with resolution. Here, given a set of clauses, S, we distribute the clauses into to sets, P and R, and in a fair way run $SInstGen$ on P and run resolution on pairs of clauses in S so long as one premise is in R. Factoring is applied to conclusions of resolution inferences. We provide a heuristic, called Ground/Single Max, for distributing the clauses of S into P and R. We also provide soundness and completeness proofs and discuss our implementation named Spectrum. Initial results identify a class of problems, LCL, that $SIGRes$ outperforms $SInstGen$ and resolution alone. We identify a few ways in which Spectrum's competitiveness can be increased.

The Completeness Proof for SIG-Res relies on ordered resolution. It may be interesting to determine if the completeness proof for SIG-Res can be extended to ordered resolution with selection and if so, how it affects the implementation's performance. Another area that might be worthy of investigating is determining for which classes of problems is SIG-Res a decision procedure and for those classes, what is the complexity?

Chapter 5

The $\Gamma + \Lambda$ Framework

Due to the common occurrence of equality in formulas of interest to the automated theorem proving community, as early as 1969, with Robinson and Wos' introduction of the paramodulation inference system [41], many have investigated calculi for first-order predicate calculus with equality. The combination of paramodulation and Knuth and Bendix's completion [54] was a significant step forward which led to the superposition[1] calculus [64] which restricts the search space for paramodulation by only requiring inferences that meet additional ordering constraints.

Recently, with provers like Vampire [102] and E [101] demonstrating the strengths of superposition, research began on combining superposition with other first-order calculi. Notable in this line of research are Lynch and Tran's paper [132] and de Moura and Bjorner's work[2][130] combining SMT with superposition and Baumgartner and Waldmann's combination of superposition with Model Evolution [140].

In this chapter we describe a sound and refutationally complete framework, called $\Gamma + \Lambda$, which allows the combination of different pairs of sound and refutationally complete calculi. We require that the two inference systems, Γ and Λ, have certain properties. First, both calculi must be sound and refutationally complete. Second, Γ must be *productive*.

[1] Throughout this chapter when we refer to superposition we mean the sound and complete inference system that includes equational factoring and equational resolution

[2] Here, SMT can be combined with any inference system with the reduction property for counterexamples

Informally[3], an inference system, Γ, is productive if when Γ is used to saturate a set of clauses, say P, we can incrementally construct a set of clauses, a candidate set, that at the limit can be used to produce a model for P when P is satisfiable. Third, Λ must have the *lifting* and *total-saturation* property[4]. Lifting is defined in the standard way [87] and total-saturation, informally, ensures that, all potential inferences are made.

In our method we first separate the input clauses into two sets [5]. The idea is, given a set of input clauses, to pre-process the clauses to determine which inference systems the clauses are best suited for. We choose two sound and refutationally complete inference systems, Γ and Λ, requiring Γ to be productive and Λ to have the lifting and total-saturation properties, and construct two sets of clauses, P_0 and R_0, by including in P_0 the clauses best suited for Γ and including in R_0 those best suited for Λ.

We then, initialize two sets $P = P_0$ and $R = R_0$ and in a fair way, apply Γ to P and apply Λ to $M \cup R$ where M is a candidate set for P. Conclusions to Γ rules and Λ rules are added to P and R respectively. Unsatisfiable cores in R are learned from, with new clauses added to P. If at any point, P is determined to be unsatisfiable, the set of input clauses is deemed unsatisfiable. Satisfiability of $P_0 \cup R_0$ is witnessed by the satisfiability of P under Γ and the total-saturation of $M \cup R$ by Λ.

Since M can be, loosely speaking, smaller than P, we can view this method as a restriction on Λ[6]. $\Gamma + \Lambda$ can also be viewed as a generalization of de Moura and Bjorner's work [130] where Γ is DPLL, Λ is superposition, P is the set of ground clauses, M is the model (set of unit literals) generated by DPLL on P and R is the set of non-ground clauses. Hence our completeness proof can be seen as a generalization of the completeness proof they provide.

Below we provide preliminary definitions in Section 5.1, discuss how to transform an

[3] A formal definition for productive is given in Section 5.1

[4] Lifting and total-saturation are also defined in Section 5.1

[5] This idea was originally suggested in [136] where we combine instance generation and resolution.

[6] M may in fact equal P, but then no benefit is gained other than Λ may witness the unsatisfiability of P quicker then Γ.

inference system into one which supports hypothetical clauses in Section 5.2, give a formal description of $\Gamma + \Lambda$ in Section 5.3 and provide proofs of soundness and completeness in Section 5.4. In Section 5.5 we show how $\mathrm{Inst\text{-}Gen\text{-}Eq}$ can be combined with superposition in this framework. In particular, in Section 5.5.1 we discuss the productive, lifting and total-saturation properties in terms of $\mathrm{Inst\text{-}Gen\text{-}Eq}$ and superposition and in Section 5.5.5 we discuss our implementation to date.

5.1 Preliminaries

Let S be a set of first-order logic formulas, renamed apart, in CNF and Γ and Λ be sound and refutationally complete first-order logic calculi.

A *distribution heuristic* can be defined as a function $\mathrm{dist} : S \rightarrow \{\{\Gamma\}, \{\Lambda\}, \{\Gamma, \Lambda\}\}$ that maps the clauses in S to a nonempty subset of $\{\Gamma, \Lambda\}$. Now let $P = \{C | \Gamma \in \mathrm{dist}(C)\}$ and let $R = \{C | \Lambda \in \mathrm{dist}(C)\}$. Clearly $S = P \cup R$.

A *candidate set* is any set (including the empty set) of clauses. Let $M_1, M_2, \cdots$ be a sequence of candidate sets. We define a *persistent candidate set*, denoted M_∞, in the following way. Let $C_1, C_2, \cdots$ be an enumeration of all the clauses in M_i for all $i \geq 1$. Let T_0 be the set of all candidate sets. For each $i \geq 1$ if C_i appears in infinitely many $M_j \in T_{i-1}$ then $T_i = \{M_j | C_i \in M_j \in T_{i-1}\}$. Otherwise, $T_i = T_{i-1}$. Now let $S_0 = \varnothing$. For each $i > 0$, if C_i is in an infinite number of T_j then $S_i = S_{i-1} \cup \{C_i\}$ otherwise $S_i = S_{i-1}$. Then we define $M_\infty = \bigcup_{i \geq 1} S_i$.

A derivation of an inference system, Γ, is a sequence, $S_0, S_1, \cdots$, of states of a system such for each $0 < i$, S_i is the result of applying a Γ inference rule on S_{i-1}. We assume all conclusions to all inferences are renamed apart. A *fair* derivation is one where all inferences are eventually performed. The persistent set, $S_\infty = \bigcup_{0 \leq ii \leq j} \bigcap S_j$.

We say that the inference system Γ is *productive* if for every fair derivation of Γ on P yielding $P_0, P_1, \cdots$ with the limit P_∞ there exists a sequence of candidate sets $M_0, M_1, \cdots$

with the limit M_∞, the persistent candidate set, such that there exists $M \subseteq Gr(M_\infty)$ where M is consistent and $M \models P_\infty$.

Given a candidate set, M, a *hypothetical clause* is a clause of the form $H \underset{\sigma}{\triangleright} C$ where H is a set of clauses in M (the hypothesis), C is a clause (the effective clause), and σ is a substitution[7]. In the case that σ is the identity, we may omit σ. A hypothetical clause, $H \underset{\sigma}{\triangleright} C$, is ground if H is a set of ground clauses, C is ground and σ is the identity. An instance of $H \underset{\sigma}{\triangleright} C$ is a hypothetical clause $H\sigma\tau \triangleright C\tau$ where τ is a substitution.

Any standard clause, K can be written as a hypothetical clause with $H = \varnothing$, $C = K$ and σ being the identity. An element K of a candidate set can be written as a hypothetical clause with $H = \{K\}$, $C = K$ and σ being the identity.

Given a set (possibly empty), S, of hypothetical clauses, we define $\mathrm{hyp}(S)$ as the union of all hypotheses of all hypothetical clauses in S. We define $\mathrm{conjoin}(S)$ as the conjunction of all clauses in $\mathrm{hyp}(S)$ and assume $\mathrm{conjoin}(S)$ to be in CNF.

Given a set of clauses V, we denote the set of all ground instances of V as $Gr(V)$. A ground clause C is *redundant* in $Gr(V)$ if there are clauses $C_1, \cdots, C_n$ in $Gr(V)$ such that $C_i \prec C$ holds for all $1 \leq i \leq n$ and $C_1, \cdots, C_n \vDash C$. A clause C is redundant in V if all of the ground instances of C are redundant in $Gr(V)$.

Let W be a set of hypothetical clauses. A ground hypothetical clause $H \triangleright C$ is redundant in $Gr(W)$ if there are $H_1 \triangleright C_1, \cdots, H_n \triangleright C_n$ in $Gr(W)$ such that C is redundant in $\{C_1, \cdots, C_n\}$ and $\bigcup_{1 \leq i \leq n} H_i \subseteq H$. A hypothetical clause K is redundant in W if all of the ground instances of K are redundant in $Gr(W)$.

We say the inference system Λ has the *lifting* property if when given a set of clauses $C_1, \cdots C_n$ and a set of ground substitutions $\sigma_1 \cdots \sigma_m$ by Λ we have $C_1\sigma_1 \cdots C_n\sigma_n \vdash C$ then by Λ we have $C_1, \cdots, C_n \vdash C'$ such that for some ground substitution τ we have $C = C'\tau$ or there exists some $0 \leq i \leq n$ such that $C = C_i\tau$. We say Λ has the *total-saturation* property if for every $T \subseteq R$ when I is a Λ inference on T then I is also an Λ

[7]Our definition is almost the same as in [130] except we include a substitution component.

inference on R.

We say a set of clauses T' is an unsatisfiable core in R if R is unsatisfiable, $T' \subset R$, and T' is a minimal set of clauses in R such that T' is unsatisfiable. If Λ produces an unsatisfiable core $T' \subset R$ then we say that Λ witnesses the unsatisfiability of R.

5.2 Transforming Inference Rules to Support Hypothetical Clauses

Inference rules can be modified in a straightforward way to support the use of hypothetical clauses. Suppose λ, depicted in Figure 5.1, is an arbitrary inference rule.

$$(\lambda) \quad \frac{D_1, \cdots, D_n}{C_1, \cdots, C_m} \quad \text{where} \quad \tau \text{ is the substitution used in the inference}$$

Figure 5.1: λ

We can construct a new inference rule λ', shown in Figure 5.2, from λ that supports hypothetical clauses as follows.

$$(\lambda') \quad \frac{H_1 \underset{\sigma_1}{\rhd} D_1, \cdots, H_n \underset{\sigma_n}{\rhd} D_n}{H \underset{\sigma}{\rhd} C_1, \cdots, H \underset{\sigma}{\rhd} C_m} \quad \text{where} \begin{cases} \text{(i)} & \tau \text{ is the substitution used in } \lambda \\ \text{(ii)} & H = \bigcup\limits_{1 \le i \le n} H_i \\ \text{(iii)} & \sigma = \sigma_1 \circ \cdots \circ \sigma_n \circ \tau \end{cases}$$

Figure 5.2: λ'

λ' is constructed by replacing clauses by hypothetical clauses and adding two new conditions. The first new condition requires that the hypothesis of each premise be added to the hypothesis of each conclusion. The second states that the substitution in the conclusion is the composition of all the substitutions from the premises with the substitution, τ, used by the inference rule. We also note that λ may have other conditions. These are added to the conditions of λ'.

5.3 $\Gamma + \Lambda$ Inference Rules

Let's again let S be a set of first-order logic formulas in CNF and Γ and Λ be sound and refutationally complete first-order logic calculi but also require that Γ be productive and Λ have both the lifting and total-saturation properties. We let P and R be sets constructed by a distribution heuristic such that $P \cup R = S$.

The inference rules for $\Gamma+\Lambda$ consists of the modified versions (discussed in Section 5.2) of the Γ and Λ inference rules plus a new inference rules called Learn and a deletion rule called Delete, given in Figure 5.3. (We note that in the definitions of the inference rules, below, we denote the set of all premises used in a rule as Prem and the set of all conclusions in a rule as Concl.)

The Γ inference rules remain unchanged aside from requiring the premises to be in P. Each Λ inference rule is however replaced with a new inference rule to support the use of hypothetical clauses. When we refer to Λ inference rules in the remainder of this chapter, we assume the rules support hypothetical clauses. One additional condition is applied to all Λ inference rules, that is, the premises are required to be in $M \cup R$ where M is the current candidate set. A hypothetical clause $(H \underset{\sigma}{\triangleright} C) \in R$ is interpreted as $H\sigma \wedge R* \models C$, where $R*$ is the set of hypothetical clauses in R that have empty hypothesis.

The Learn inference rule adds a new clause to P which witnesses the inconsistency of $M \cup R$. If the inconsistency involves M, the learned clause refines the next candidate model. If not, the learned clause signals the original set of clauses is unsatisfiable.

Given a candidate set, M, Delete removes from R the hypothetical clauses that are not implied by M, that is, given a hypothetical clause $H \underset{\sigma}{\triangleright} C \in R$, if $H \not\subseteq M$ then $H \underset{\sigma}{\triangleright} C$ is removed from R. The Delete rule given in Figure 5.3 is a state transition diagram. $e_1|e_2|e_3$ denotes a state of the $\Gamma + \Lambda$ system where e_1 is the current candidate model, $e_2 = P$ and $e_3 = R$. We note that Delete is not required for completeness.

A $\Gamma + \Lambda$ *derivation* from S is a sequence $(P_0, R_0, M_0), \cdots, (P_i, R_i, M_i), \cdots$, where

$$\text{Learn} \quad \frac{H_1 \underset{\sigma_1}{\triangleright} C_1, \cdots, H_n \underset{\sigma_n}{\triangleright} C_n}{\neg(H\sigma)} \quad \text{where} \begin{cases} \text{(i)} & \text{Prem} \subseteq M \cup R \\ \text{(ii)} & \{C_1, \cdots, C_n\} \text{ witnesses unsatisfiability of } M \cup \\ \text{(iii)} & H = \text{conjoin}(\text{Prem}) \\ \text{(iv)} & \sigma = \sigma_1 \circ \cdots \circ \sigma_n \end{cases}$$

$$\text{Delete} \quad \frac{M|P|R', H \underset{\sigma}{\triangleright} C}{M|P|R'} \quad \text{where} \quad H \not\subseteq M$$

Figure 5.3: Learn and Delete Inference Rules

1. $P_0 \cup R_0 = S$,
2. for all $i \geq 0$ P_i, R_i and M_i are multisets of clauses where M_i is a candidate set for P_i and
3. for all $i \geq 0$ one of the following holds: (a) P_{i+1} results from applying a Γ inference rule or Learn rule on P_i and $R_{i+1} = R_i$ or (b) R_{i+1} results from applying a Λ inference rule or Delete rule on R_i and $P_{i+1} = P_i$.

A $\Gamma + \Lambda$ derivation from S has as its limit the sets of *persistent clauses* $P_\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} P_j$ and $R_\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} R_j$ and a persistent candidate set M_∞. By definition of redundancy, if a clause is redundant in some P_i it is redundant in P_∞ and if a hypothetical clause, $H \underset{\sigma}{\triangleright} C$, is redundant in some R_i and $H\sigma \subseteq M_\infty$ then it is redundant in R_∞.

Let $(P_\infty, R_\infty, M_\infty)$ be the limit of a derivation. P_∞ is called *saturated* if the conclusion of every Γ inference and Learn inference is in P_∞ or is redundant in P_∞. R_∞ is saturated if the conclusion of every Λ inference is in R_∞ or is redundant in R_∞. A derivation is *fair* if no inference is ignored forever, i.e. the conclusion of every inference among persistent clauses is persistent or redundant in (P_∞, R_∞). A fair derivation produces saturated sets.

Unsatisfiability of S under $\Gamma + \Lambda$ is detected by Γ witnessing the unsatisfiability of P_∞. Satisfiability is detected by a saturated system without Γ witnessing the unsatisfiability of P_∞.

A saturation procedure for $\Gamma + \Lambda$ is provided in Algorithm 6. Here $\Gamma(P, m)$ returns the set P after the application of m steps in the saturation of P with Γ inference rules and $\text{Delete}(M, R)$ returns the set R after the deletion of the clauses in R not implied by the

candidate set M. $\Lambda(M \cup R, n)$ returns the set R after the application of n steps in the saturation of $M \cup R$ with Λ inference rules and given an unsatisfiable core, K, $\mathrm{Learn}(K)$ returns a set containing the conclusion(s) to the Learn inference rule with the clauses in K as the premises.

Algorithm 6: $\Gamma + \Lambda$

input : Sets P and R containing FO formulas with equality in CNF
output: SATISFIABLE or UNSATISFIABLE

Let m, n be positive integers.
while *P and R are not saturated* **do**
  $P := \Gamma(P, m)$;
  if *P is unsatisfiable witnessed by Γ* **then**
    return UNSATISFIABLE;
  Construct candidate set M from P;
  $R := \mathrm{Delete}(M, R)$;
  $R := \Lambda(M \cup R, n)$;
  if *$M \cup R$ is unsatisfiable witnessed by Λ* **then**
    foreach *Unsatisfiable core, $K \subseteq M \cup R$* **do**
      $P := P \cup \mathrm{Learn}(K)$;
return SATISFIABLE;

5.4 Soundness and Completeness

Our proofs of soundness and completeness of $\Gamma + \Lambda$ is a generalization of the proofs of the soundness and completeness of $DPLL(\Gamma)^\sharp$ given in [130].

Let $S = P \cup R$ be a multiset of clauses and let Γ and Λ be sound and refutationally complete inference systems where Γ is productive and Λ has the lifting and total-saturation properties. Since Γ and Λ are sound inference systems it is clear that all Γ and Λ inferences in $\Gamma + \Lambda$ preserve satisfiability of S. Learn is sound since Learn produces a clause stating that at least one of the clauses, in a set of hypotheses that produce an unsatisfiable core, is not true. Delete is sound since it removes hypothetical clauses that are no longer implied by the current candidate set.

Theorem 8 $\Gamma + \Lambda$ *is sound.*

Theorem 9 *Let* $S_0 = P_0 \cup R_0$ *be a multiset of clauses and let* Γ *and* Λ *be sound and refutationally complete calculi with* Γ *being productive and* Λ *having the lifting and total-saturation property. If* $\{P_\infty, R_\infty, N_\infty\}$ *is the limit of a derivation of* $\Gamma + \Lambda$ *on* S_0 *and* Γ *does not witness the unsatisfiability of* P_∞ *then* S_0 *is satisfiable.*

Proof Let $S_0 = P_0 \cup R_0$ be a multiset of clauses and let Γ and Λ be sound and refutationally complete calculi where Γ is productive and Λ has the lifting and total-saturation properties. Suppose $\{P_\infty, R_\infty, N_\infty\}$ is the limit of a derivation of $\Gamma + \Lambda$ on S_0 and suppose Γ does not witness the unsatisfiability of P_∞. Let P,R, and N be the set of ground instances of P_∞, R_∞ and N_∞, respectively. As Γ is productive and P_∞ is satisfiable, there exists $M \subseteq N$ such that M is consistent and $M \models P$. Let $R_M = \{H \triangleright C \in R | H \subseteq M\}$.

We claim $M \cup R_M$ is saturated by Λ, that is, the conclusion of every Λ inference on $M \cup R_M$ is in R_M or is redundant in R_M. Suppose on the contrary that there exists some clause $K = H \triangleright C$ that is the result of a Λ inference rule on $M \cup R_M$ which is not in R_M and is not redundant in R_M. Since K is the result of a Λ inference rule on $M \cup R_M$ we have $H \subseteq M$.

Since $N_\infty \cup R_\infty$ is saturated by Λ and Λ has both the lifting property and the total-saturation property, then $K \in R$ or is redundant in R. Suppose $K \in R$. Then as $K \notin R_M$, it follows that $K \in R \setminus R_M$. Hence $H \not\subseteq M$, a contradiction. Now suppose that K is redundant in R. Then there are $H_1 \triangleright C_1, \cdots, H_n \triangleright C_n$ in R such that C is redundant in $\{C_1, \cdots, C_n\}$ and $\bigcup_{1 \leq i \leq n} H_i \subseteq H$. Since K is not redundant in R_M, $H_j \triangleright C_j \notin R_M$ for some $1 \leq j \leq n$. Therefore $H_j \not\subseteq M$ which implies $H \not\subseteq M$, a contradiction. Hence our claim that $M \cup R_M$ is saturated by Λ is proven.

Next we claim that $M \cup R_M$ is satisfiable and will show equivalently that $M \cup R_M$ does not contain an unsatisfiable core. Suppose $M \cup R_M$ does contain an unsatisfiable core, $D = H_1 \triangleright C_1, \cdots, H_m \triangleright C_m$. Set $H = \bigcup_{1 \leq i \leq m} H_i$. If $H = \varnothing$ then by Learn, $\square \in P_\infty$. But

P_∞ is satisfiable, hence a contradiction. If $H \neq \varnothing$, then $M \models \mathrm{conjoin}(H)$. By Learn, $\neg(\mathrm{conjoin}(H)) \in P_\infty$ or is redundant in P_∞. Thus $M \models \neg(\mathrm{conjoin}(H))$, a contradiction. Hence $M \cup R_M$ is satisfiable.

Since $M \models P_\infty$ we have $M \models P_0$. Since for all $C \in R_0$, C contains an empty hypothesis and by our definition of redundancy it follows that $R_M \models R_0$. Thus, since $M \cup R_M$ is satisfiable, S_0 is satisfiable. ∎

5.5 Combining Inst-Gen-Eq and Superposition

The only work, that we are aware of, that combines SInst-Gen and superposition is the system by Ganzinger and Korovin called Inst-Gen-Eq[111]. In their work they provide a calculus which extends SInst-Gen with superposition to support formulas that include equality[8].

In their method, the role of superposition is to generate a proof of inconsistency in the current candidate model. When a proof of an inconsistency is found, the substitution used in the proof is applied to the clauses whose literals are in the proof via SInst-Gen. These new instances help to refine the next candidate model.

We believe, their method can be enhanced in our framework so that the full power of superposition can be utilized. In this section we show how, using the $\Gamma + \Lambda$ framework, Inst-Gen-Eq and superposition can be combined in a way that takes full advantage of each of the two calculi. We call this inference system SIG-Sup.

5.5.1 Γ and Λ Properties

Among the requirements of the $\Gamma + \Lambda$ framework is that the inference system Γ must be productive and Λ must have the lifting and total-saturation properties. Here we show that

[8]Later in [142],Korovin and Sticksel describe their implementation *iProver-Eq* which implements the calculus given in [111]

$InstGenEq$ is productive and discuss the lifting and total-saturation properties of superposition.

For the sake of convenience, we provide definitions for terms that are used in this section which were defined previously in Chapter 4. $\perp$ is used to denote a distinguished constant called the *Jeroslow constant* and ambiguously the substitution which maps all variables to $\perp$. If L is a literal then $L\perp$ denotes the ground literal obtained by applying the $\perp$-substitution to L and if P is a set of clauses then $P\perp$ denotes the set of ground clauses obtained by applying the $\perp$-substitution to the clauses in P.

$P\perp$ can be viewed as a set of propositional clauses. Under this setting, if $P\perp$ is unsatisfiable, then P is unsatisfiable. Otherwise a model for $P\perp$ is denoted as $I_\perp$. We define a selection function, $\mathrm{sel}(C, I_\perp)$, which maps each clause $C \in P$ to a singleton set $\{L\}$ such that $L \in C$ and $L\perp$ is true in $I_\perp$. We define the candidate set $I_P = \bigcup_{C \in P} \mathrm{sel}(C, I_\perp)$.

During the $\mathrm{Inst\text{-}Gen\text{-}Eq}$ saturation of a set of clauses, P, a set of ground instances of P, denoted $P_\perp$, is sent to an SMT solver. If $P_\perp$ is determined unsatisfiable, P is determined unsatisfiable and we are done. Otherwise, a set of literals, denoted $I_\perp$, where $I_\perp \models P_\perp$, is retrieved from the SMT solver. We then construct I_P, a candidate model, from $I_\perp$ and P using the selection function. We then determine, using unit paramodulation, if I_P is consistent. If so, then P is satisfiable, as I_P can be used to generate a model of the ground instances of P, and we are done. Otherwise, if I_P is found to be unsatisfiable, the substitution used in the proof of unsatisfiability is used to generate new instances of the clauses (via $\mathrm{SInst\text{-}Gen}$) whose selected literals are in the proof. These new instances help refine the next candidate model. The process is repeated until $P_\perp$ is unsatisfiable or there exists an I_P that is consistent.

In this setting, since for each ground instance, one literal is selected in each I_P and since there are finitely many literals in a clause, one literal will be chosen infinitely often. Hence $\mathrm{Inst\text{-}Gen\text{-}Eq}$ is productive.

It is well-known in the literature that superposition can be lifted from the ground case

[90]. And since we can exhaustively apply superposition on all subsets of R, even in the presence of the empty clause, superposition has the total-saturation property.

5.5.2 The SIG-Sup Inference System

Under the $\Gamma + \Lambda$ framework, let Γ be $InstGenEq$ and let Λ be superposition. Let S be a set of first-order formulas in CNF. Let P and R be sets of clauses such that $S = P \cup R$. Let M be a candidate set for P. We define two selection function, one for Inst-Gen-Eq inferences and another for our modified superposition inferences.

We define $\mathrm{sel}_P(C, I_\perp)$, where $C \in P$ and $I_\perp$ is a model for $P\perp$, as follows.

$$\mathrm{sel}_P(C, I_\perp) = \{L\} \text{ for some } L \in C \text{ such that } L\perp \in I_\perp$$

We define $\mathrm{sel}_{M\cup R}(C)$, where $C \in M \cup R$ as follows.

$$\mathrm{sel}_{M\cup R}(C) = \max(C)$$

For clarity, we note that $\mathrm{sel}_P(C, I_\perp)$ returns a singleton set and $\mathrm{sel}_{M\cup R}(C)$ returns a non-empty set. The inferences rules for SIG-Sup are given in Figures 5.4.

Since both Inst-Gen-Eq and superposition are sound and refutationally complete, Inst-Gen-Eq is productive and superposition has the lifting and total-saturation properties, by the soundness and completeness of $\Gamma + \Lambda$ it follows that SIG-Sup is both sound and complete.

5.5.3 SIG-Sup Saturation Process

The system we propose takes as input a first order formula (with equality) in CNF, S, and first, by some heuristic, establishes two sets of clauses, P and R, such that P contains clauses that are better suited for Inst-Gen-Eq and R contains clauses that are better suited for superposition. Any heuristic can be used so long as $P \cup R = S$. In [136] we proposed

a heuristic we called, Ground-Single Max (GSM) where P contains ground clauses and clauses containing more than one maximal literal and R all other clauses. Although the division heuristic does not alter the system's completeness, it can obviously affect performance.

After the clauses have been distributed, the system starts in search mode[9]where we first check to see if P, when viewed as a set of ground formula, is unsatisfiable. This is a quick check. In order to do this, we *jeroslowize* P, replacing all variables in P with the Jeroslow constant, traditionally denoted $\bot$, to produce $P\bot$. We check the satisfiability of $P\bot$ using SMT, possibly adding additional clauses, G, to P in the process to rule out inconsistencies in the ground formula. If $P\bot$ is unsatisfiable we halt and we conclude that S is unsatisfiable.

If on the other hand, $P_\bot$ is satisfiable, we switch to saturation mode and construct a set of *selected literals*, I_P. That is, for each clause $C \in P$ we choose a single literal L in C such that $L_\bot \in I_\bot$ and add it to I_P. Clearly I_P, our candidate set, implies P.

Then given I_P and R, we first remove any hypothetical clauses in R that contain literals in their hypotheses that are not in I_P via Delete inferences. Then, in a fair way, we saturation $I_P \cup R$ with superposition, equality resolution and equality factoring inferences, adding all conclusions to R.

In $\mathrm{SIG\text{-}Sup}$, unsatisfiability of $I_P \cup R$ is detected by a single clause, $H \underset{\sigma}{\triangleright} \Box \in R$. If such a clause is produced we enter conflict resolution mode. In conflict resolution mode we analyze the conflict to determine a course of action. If $H = \varnothing$, then the initial set of clauses R is unsatisfiable and we report that S is unsatisfiable and halt. (Note that this is equivalent to adding the empty clause to P via Learn). If $H \underset{\sigma}{\triangleright} \Box$ can be derived from I_P only, then there is a conflict in I_P, the candidate model. Here the learned clause is a tautology which need not be added to P. But we do add to P new instances (using $\mathrm{SInst\text{-}Gen}$) of the clauses whose selected literals are in H and go back into search mode directing the SMT solver

[9]This is similar to the search and conflict resolution modes used in [130]. In our system we differentiate between search, saturation, and conflict modes.

to again check the consistency of $P_\perp$. The substitution used in the SInst-Gen inference is the substitution used in the proof of inconsistency of I_P. If I_P conflicts with R we apply the Learn inference rule to $H \underset{\sigma}{\triangleright} \square$, adding a new clause to P, and re-enter search mode. If $I_P \cup R$ becomes saturated and R does not contain the empty clause, we conclude that S is satisfiable and halt.

We note that the $\Gamma + \Lambda$ framework requires that (i) P be be saturated by Inst-Gen-Eq, thus requiring the saturation of the *candidate model* by superposition and (ii) that the *candidate set* union R be saturated by superposition. Since the candidate model, I_P, used in Inst-Gen-Eq is exactly our candidate set, doing these saturation processes separately would result in I_P being saturated by superposition twice. In the method we propose, inconsistencies in the candidate model/set, I_P, are found during the saturation of $I_P \cup R$ by superposition. These inconsistencies trigger SInst-Gen inferences, resulting in the saturation of P by Inst-Gen-Eq, without the duplication of effort.

We believe that this combination produces a more efficient system than previously suggested systems by (i) identifying a set of clauses, P, which is better suited for Inst-Gen-Eq and a set, R, which is better suited for superposition, (ii) detecting ground unsatisfiability of P efficiently using SMT, (iii) controlling the size of R by eliminating hypothetical clauses that are not implied by the current interpretation and (iv) minimizing the size of the clauses in R by applying superposition on $I_P \times R$ rather than $P \times R$.

5.5.4 SIG-Sup Algorithm

An algorithm for applying the inferences in SIG-Sup in a fair way is given in Algorithm 7. Note that we do not follow Algorithm 6. The main difference is that we guide the saturation of P with SInst-Gen using the inconsistencies in I_P detected in the saturation of $I_P \cup R$ with superposition. This has the effect of saturating P with Inst-Gen-Eq and $I_P \cup R$ with superposition while avoiding running superposition on I_P twice. Since superposition has the total-saturation property we are assured that any inconsistency in I_P will be found,

resulting in $\mathrm{SInst\text{-}Gen}$ inferences being added to P.

We define the functions in Algorithm 7 as follows. $\mathrm{SMT}(P)$ takes a set of clauses, P, as an argument and returns a set of clauses. $\mathrm{SMT}(P)$ generates a ground abstraction of P, denoted $P_\perp$, and determines the satisfiability of $P_\perp$. If $P_\perp$ is determined to be unsatisfiable, $\mathrm{SMT}(P)$ returns $\varnothing$, otherwise $\mathrm{SMT}(P)$ returns a set of clauses (often unit) that entail $P_\perp$. $\mathrm{Superposition}(I_P \cup R)$ takes a set of clauses $I_P \cup R$, where I_P is the current candidate set as input and returns a set of clauses. $\mathrm{Superposition}(I_P \cup R)$ attempts to saturate $I_R \cup R$ with superposition, in a fair way. If at anytime, R contains the empty clause or $M \cup R$ becomes saturated, the function halts and returns the set R. $\mathrm{Inst\text{-}Gen}(P, \sigma)$ takes as input a set of clauses P and returns a set of instances, using the substitution σ, of the clauses in P. $\mathrm{Learn}(H, \sigma)$ returns a singleton set containing the conclusion to the Learn inference rule.

Algorithm 7: SIGSup(P,R)

input : Sets P and R containing FO formula with equality in CNF
output: SATISFIABLE or UNSATISFIABLE

Let $I_\perp$, I_P be sets of clauses.

while $true$ **do**
- $I_\perp := \mathrm{SMT}(P)$;
- **if** $I_\perp = \varnothing$ **then**
  - **return** UNSATISFIABLE;
- $I_P := \{L | L = \mathrm{sel}(C, I_\perp), \forall C \in P\}$;
- $R := \mathrm{Superposition}(I_P \cup R)$;
- **if** $H \overset{\sigma}{\triangleright} \square \in R$ **then**
  - **if** $H = \varnothing$ **then**
    - **return** UNSATISFIABLE ;
  - **else if** $I_P \vdash H \overset{\sigma}{\triangleright} \square$ **then**
    - $P = P \cup \mathrm{Inst\text{-}Gen}(P, \sigma)$;
  - **else**
    - $P = P \cup \mathrm{Learn}(H, \sigma)$;
- **else**
  - **return** SATISFIABLE ;

5.5.5 EVC3

EVC3, which is still under development, is our attempt to combine the strengths of existing software systems to solve the first-order validity problem using SIG-Sup. The current version is a sound and complete theorem prover that implements the $SIGRes$ inference system.

In EVC3 we couple together the SMT solver CVC3 [124] and the purely equational prover E [112]. We chose CVC3 and E for a number of reasons. First, because both systems are open source, we can offer the source code of our version freely to the public. Second, both systems are well known and have reputations for their strong performance and usability in the SMT and ATP communities respectively. Third, by using existing code bases, we eliminate the need to "reinvent the wheel".

In developing EVC3, special care was taken as to not alter any existing code in CVC3, but to utilize the facilities and architecture of CVC3 and augment its code base, initially with an implementation of the SIG-Res inference system. To execute the system utilizing our new code the command line flag `+sig-res` is used. Otherwise, EVC3 behaves identically to CVC3. The only code modified in E was its main function to facilitate the creation of an API for its incremental use.

The source code is maintained under a single directory structure and can be configured and built using a single configure script and make command. The executable runs in a single process without threading.

A high level view of EVC3 can be found in Figure 5.5. The components defined by solid bold lines are completely new. Components defined by bold broken lines have been augmented to support running CVC3 and E together using our quantifier theory reasoner.

At first, CVC3 initializes all of its subsystems; i.e. SAT Solver, DPLL Engine, Search Engine, and Theory Core (solver). When EVC3 is run with the `+sig-res` flag, CVC3 installs our new quantifier theory decision procedure which implements the SIG-Res inference rules and CVC3 installs and initializes E.

After the subsystems are initialized, a problem is read into the system, either interactively via the command line or from a file. A new parser was added to CVC3 which allows problems in TPTP3 [79] format to be read into the system, which is done using the command line option `+lang tptp`.

When a problem is being read into the system, a distribution heuristic is applied to the individual clauses to determine if the clause should be sent to E by which it is added to R, or should remain in CVC3 and added to P. There are currently three distribution heuristics built into EVC3 and can be chosen via the command line flag `+dist-mode mode`. The built in modes are `ig-only` (all clauses are added to P), `res-only` (all clauses are added to R), and the default mode `gsm` (defined above). Maximality in the GSM heuristic is determined via a Knuth-Bendix ordering.

Once the formulas are distributed, CVC3 continually executes a loop in its SAT solver, checking the satisfiability of $P_\perp$. If it determines $P_\perp$ unsatisfiable it halts and reports unsatisfiable. Otherwise if satisfiable, it queries the Theory Solver for the consistency of the model. If the Theory Solver (which calls the individual theory decision procedures) determines the model to be inconsistent, new formulas explaining the inconsistency are passed back to the SAT solver and the SAT solver continues in its loop. If no inconsistency is found by the Theory Solver, CVC3 halts and reports satisfiable.

SIG-Res Quantifier Theory Module

Our quantifier theory decision procedure in the SIG-Res Quantifier Theory Module maintains the set of clauses P and retrieves the clauses in R from E when needed.

We have added to CVC3 a new distinguished constant, $\perp$, similar to CVC3's Skolem constant. For each formula ϕ that is added to P the quantifier theory module passes $\phi_\perp$ to the SAT solver.

As noted above, when the quantifier theory decision procedure is queried for consistency, at that point the set $P_\perp$ is satisfiable.

The quantifier theory decision procedure then applies the inference rules of SIG-Res in a fair manner until one of the following three states occurs, at which time it passes control back to the SAT solver.

1. It deduces the empty clause, implying the original set of clauses, F, is unsatisfiable. When this occurs the quantifier decision procedure adds $\bot$ to $P_\bot$ forcing the SAT solver to report unsatisfiable.
2. It concludes that F is saturated with respect to SIG-Res, implying F is satisfiable, and returns nothing.
3. It returns at least one new clause $\phi_\bot$ not in $P_\bot$.

Resolution and factoring inferences on clauses in R are delegated to E, Whereas all inferences involving clauses in P are performed directly in the quantifier decision procedure.

E API

A new API was created to allow CVC3 and E to communicate and exchange data between their systems. The E API, found in the class `SPSolver`, allows a user to initialize E, add clauses to E's initial set of clauses, run E for a fixed number of steps of its saturation process, query E's state, retrieve the set of clauses processed by E, and gracefully terminate E.

Implementation Status

The current version of EVC3 is a sound and complete theorem prover for first-order formula without equality. In EVC3, we currently support the $SIGRes$ inference system and are working to extend it to support $SIGSup$.

As EVC3 is new and still under development, it is not yet competitive with state-of-the-art theorem provers. Before it can complete with the more mature provers we must address a number of issues. The major work to be done is to augment the parser and quantification

theory to allow formula with equality and to modify the algorithm in the quantification theory to implement SIG-Sup.

We would like to include the ability to input arguments for E on the command line rather than having the arguments hard coded in the initialization of E. This will allow the user to utilize the command line flags of E. Also, a tighter coupling of E to CVC3 by creating methods to convert E data to CVC3 data, and vice versa, will improve efficiency. We would like to extend our quantifier decision procedure to handle full first order logic formula rather than just CNF formulas. More work also needs to go into the quantifier reasoner to eliminate redundancy when performing inferences on clauses in P and restricting the search space possibly by using *dismatching constraints* in the SInst-Gen procedure. We also would like to investigate heuristics for determining i) the number of new inference clauses to add to P before returning from a SAT query, ii) the balance between inferences on clauses in P and steps taken in E's saturation process and iii) the distribution of clauses to P and R.

5.6 Conclusion

In this chapter we presented a framework called $\Gamma + \Lambda$ which allows different combinations of inference systems to be combined into a single sound and complete system. The only requirements on the inference systems are that they both be sound and refutationally complete and that one be productive and the other have both the lifting property and the total-saturation property.

In presenting this framework we provide transformation rules which allow an inference system to support hypothetical clauses, then give the Learn and Delete inference rules, and follow up with soundness and completeness proofs.

We show how, under this framework, we can combine Inst-Gen-Eq with superposition into a single sound and refutationally complete system. We call our system SIG-Sup. We

provide the inference rules for SIG-Sup, give an informal description of the system and provide an algorithm that can be used to implement such a system.

Future work consists of extending EVC3 to support SIG-Sup. Other problems of interest are whether superposition can be used as Γ, in particular, whether we can establish that superposition is productive. It would also be interesting to consider the use of $\Gamma + \Lambda$ in distributive computing. Since *small* amounts of data are exchanged between Γ and Λ, our framework may provide a new paradigm for this area of research. Lastly, as with all inference systems of practical use, redundancy criteria in our framework should be investigated.

Superposition

$$\frac{J \underset{\sigma'}{\triangleright} L[u]_p \vee \Sigma \qquad K \underset{\sigma''}{\triangleright} l \simeq r \vee \Delta}{H \underset{\sigma}{\triangleright} (L[r]_p \vee \Sigma \vee \Delta)\tau} \text{ if } \begin{cases} \tau = mgu(u, l) \\ u \textit{ is not a variable} \\ L \textit{ is of the form } s[u]_p \simeq t \textit{ or } s[u]_p \not\simeq t \\ t\tau \not\geq s[u]\tau \textit{ and } r\tau \not\geq l\tau \\ (L)\tau \in \mathrm{sel}_{M \cup R}((L \vee \Sigma)\tau) \\ (l \simeq r)\tau \in \mathrm{sel}_{M \cup R}((l \simeq r \vee \Delta)\tau) \\ L[u]_p \vee \Sigma \in M \cup R \textit{ and } l \simeq r \vee \Delta \in M \cup R \\ (L[r]_p \vee \Sigma \vee \Delta)\tau \in R \\ H = J \cup K \\ \sigma = \sigma' \circ \sigma'' \circ \tau \end{cases}$$

Equality Resolution

$$\frac{H \underset{\sigma'}{\triangleright} l \not\simeq r \vee \Delta}{H \underset{\sigma}{\triangleright} (\Delta)\tau} \text{ if } \begin{cases} \tau = mgu(l, r) \\ (l \not\simeq r)\tau \in \mathrm{sel}_{M \cup R}((l \not\simeq r \vee \Delta)\tau) \\ H \underset{\sigma'}{\triangleright} l \not\simeq r \vee \Delta \in M \cup R \\ \sigma = \sigma' \circ \tau \\ H \underset{\sigma}{\triangleright} (\Delta)\tau \in R \end{cases}$$

Equality Factoring

$$\frac{H \underset{\sigma'}{\triangleright} l \simeq r \vee s \simeq t \vee \Delta}{H \underset{\sigma}{\triangleright} (l \simeq t \vee r \not\simeq t \vee \Delta)\tau} \text{ if } \begin{cases} \tau = mgu(l, s) \\ r\tau \not\geq l\tau \textit{ and } t\tau \not\geq s\tau \\ (l \simeq r)\tau \in \mathrm{sel}((l \simeq r \vee s \simeq t \vee \Delta)\tau) \\ H \underset{\sigma'}{\triangleright} l \simeq r \vee s \simeq t \vee \Delta \in M \cup R \\ \sigma = \sigma' \circ \tau \\ H \underset{\sigma}{\triangleright} (l \simeq t \vee r \not\simeq t \vee \Delta)\tau \in R \end{cases}$$

SInst-Gen

$$\frac{L \vee \Gamma}{(L \vee \Gamma)\sigma} \text{ if } \begin{cases} H \underset{\sigma}{\triangleright} \square \in R \\ I_P \vdash H \underset{\sigma}{\triangleright} \square \\ L \vee \Gamma \in P \\ L \in \mathrm{sel}_P(L \vee \Gamma) \\ \{L\} \in H \subseteq I_P \\ (L \vee \Gamma)\sigma \in P \end{cases}$$

Learn

$$\frac{H \underset{\sigma}{\triangleright} \square}{\neg(H\sigma)} \text{ if } \begin{cases} H \underset{\sigma}{\triangleright} \square \in R \\ I_P \not\vdash H \underset{\sigma}{\triangleright} \square \neg(H\sigma) \in P \end{cases}$$

Delete

$$\frac{I_P | P | R', H \underset{\sigma}{\triangleright} C}{I_P | P | R'} \text{ if } H \not\subseteq I_P$$

Figure 5.4: SIG-Sup Inference Rules

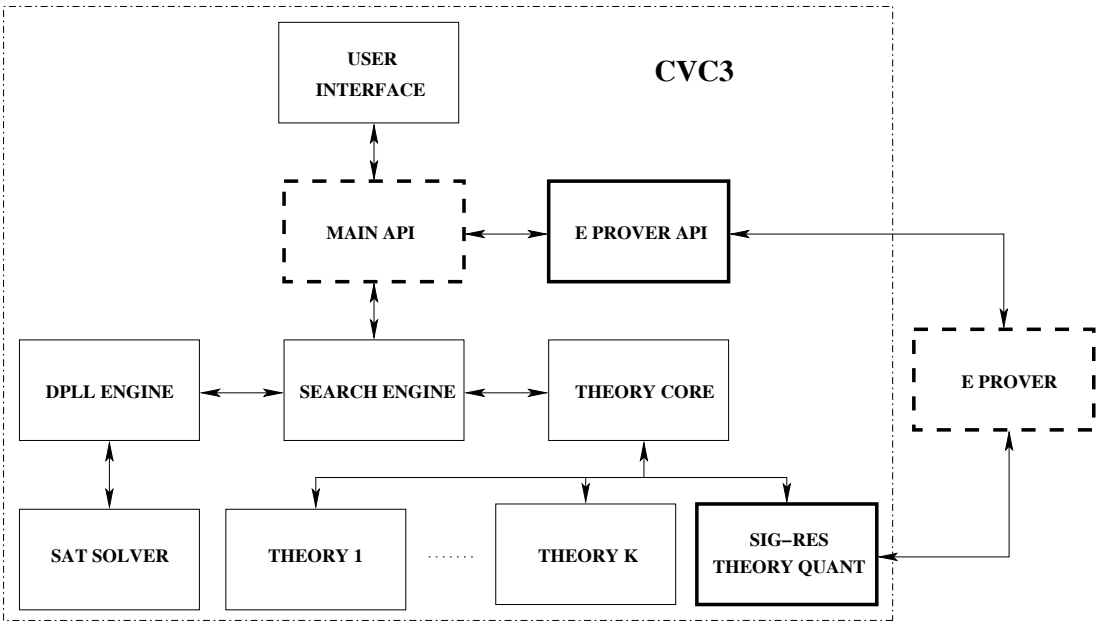


Figure 5.5: EVC3 System Diagram

Chapter 6

Conclusion

The automation of proving theorems in first-order predicate calculus is a relatively young and active field of research today with many interesting problems. In the early days, a great deal of interest was in instance-based inference systems based on Herbrand's theorem [21]. One ground breaking result from this line of research was the Davis-Putnam-Logeman-Loveland (DPLL) procedure [29, 34] which revolutionized the solving of problems in SAT. Soon after however, Robinson presented his resolution rule [36] which changed the focus of research in the community. Rather than trying to reduce first-order logic unsatisfiability to propositional logic unsatisfiability, resolution seeks to find a proof of the empty clause (falsum). Following up his work on resolution, Robinson joined up with Wos to derive paramodulation [41] which along with resolution and factoring form a refutationally complete inference system for first-order logic *with equality*. Near the same time that paramodulation was developed Knuth and Bendix developed a rewriting system for unit equations called Knuth-Bendix completion [44]. The combination of Knuth-Bendix completion and paramodulation by Brand established a restriction on the paramodulation inferences necessary for completeness which led to the superposition calculus by Zhang, Kapur, Bachmair and Ganzinger [62, 64] which is used in the fastest modern day theorem provers.

As the computational power of computers advanced in the 1980's, SAT solvers became

able to solve problems containing tens of thousands of variables and clauses. This renewed interest in instance generation-based systems. Jeroslow [63] determined that new ground instances could be generated by considering only pairs of conflicting literals from different clauses and that any variable not involved in the conflict could simply be mapped to a distinguished constant. He called this partial instantiation. Hooker, Ragu, Chandru and Shrivastava then formulated the first refutationally complete partial instantiation method in their Primal method [99] and Ganzinger and Korovin formalized primal partial instantiation in their semantic instance generation rule, SInst-Gen, and proved its completion [106]. iProver [142], a state of the art instance generation prover, is one of the most competitive theorem provers in competition which shows that although resolution based systems are at the top of the heap, it may be only a matter of time before instance-based systems are shown to be equally or perhaps more effective for theorem proving.

We discussed three novel approaches for solving the first-order validity problem using SAT. The first reduces first-order validity to propositional satisfiability. The second establishes a method to combine SInst-Gen with resolution. The third provides a general framework for combining different inference systems into a single system.

A first line of research involved establishing unsatisfiability in first-order logic by encoding the existence of a first order proof in propositional logic and utilizing state of the art SAT solvers to determine the satisfiability of the encoding. Working with DeShane, Hu, Jablonski, Lin and Lynch, we established a SAT encoding of rigid closed connection tableaux proofs, showed how it can be used for general first-order theorem proving, and proved its soundness and completeness [126]. We also identified, with our ChewTPTP-SAT implementation, problems that our system was able to solve but others could not. Though not currently competitive, we have reason to believe that by eliminating restarts in the implementation, the solver will be much more competitive.

Following this work was joint work with Bongio, Katrak, Lin and Lynch where we established a closed rigid connection tableaux proof in SMT [128]. In this encoding we

encoded the choices made in constructing the connection tableaux in SAT and encoded the unification checks and finiteness of the tableaux in SMT. In an implementation for this encoding, called ChewTPTP-SMT, we found that the encodings were significantly smaller in the Horn case and hence faster than the SAT encoding, and found that in the non-Horn case that the results were worse. Similar to ChewTPTP-SAT, in order to be competitive with state-of-the-art theorem provers, more work needs to be done on the implementation. Eliminating restarts and finding a way to reduce the non-Horn encoding by a factor of n are initial improvements that can be made. Other future work on the tableaux encoding method includes looking at ways to be able to use SMT to further reduce the representation for non-Horn clauses, ideally cutting it down to a quadratic number of clauses. In addition, in order to prove the general first order problem we also need to find a good heuristic to decide exactly which clauses should be copied. We would like a method to decide satisfiability from rigid satisfiability. It would be useful to have an encoding of rigid clauses modulo a non-rigid theory, as discussed in [122]. This way, we could immediately identify some clauses as non-rigid, and work modulo those clauses.

In the second line of research, in collaboration with Lynch [136], we developed a refutationally complete inference system called $\mathrm{SIG\text{-}Res}$ which combines semantic selection instance generation and resolution. In this system, we create two sets of clauses, R and P, and only allow resolution inferences between pairs of clauses where at least one clauses is in R. New instances of clauses in P are generated using $\mathrm{SInst\text{-}Gen}$. We established the soundness and completeness of $\mathrm{SIG\text{-}Res}$ and have demonstrated in our implementation, named Spectrum, that we are able to solve some problems faster using $\mathrm{SIG\text{-}Res}$ than using $\mathrm{SInst\text{-}Gen}$ or resolution alone. Though sound and complete, a lack of maturity, has hindered its performance relative to state-of-the-art theorem provers. More work needs to go into redundancy elimination in order for it to compete with the leading solvers. A notable success of the implementation is the identification of a number of problems in the LCL class of TPTP problems which were solved faster using $SIGRes$ than when run with

$SInstGen$ or resolution alone. We believe these problems work best using $SIGRes$ because the heuristic places *growing* clauses in R and clauses with a transitive like structure in P.

The last line of research we discuss is a framework, called $\Gamma + \Lambda$, which allows the combination of two inference systems, Γ and Λ. The requirements for Γ and Λ are that Γ and Λ be refutationally complete, Γ be productive, and Λ have the lifting and total-saturation property. Any Γ and Λ with these properties can be combined under our framework into a single sound and refutationally complete system. We present the inference rules for Γ and Λ, prove its completeness and show how under this framework that $\mathrm{Inst\text{-}Gen\text{-}Eq}$ and superposition can be combined into a single sound and refutationally complete system. We discuss our work in progress, called EVC3, which will eventually be extended to support $\mathrm{SIG\text{-}Sup}$. The current version, though sound and complete, only implements the $\mathrm{SIG\text{-}Res}$ inference system.

Other problems of interest related to the $\Gamma + \Lambda$ framework are whether superposition can be used as Γ, in particular, whether we can establish that superposition is productive. It would also be interesting to consider the use of $\Gamma + \Lambda$ in distributive computing. Since *small* amounts of data are exchanged between Γ and Λ, our framework may provide a new paradigm for this area of research. Lastly, as with all inference systems of practical use, redundancy criteria in our framework should be investigated.

Bibliography

[1] Aristotle. Prior Analytics. In A.J. Jenkinson, trans., *Internet Classics Archive*. Originally written in 350 B.C.E.

[2] P. Abaelardus. Dialectica. In L.M. de Rijk, trans., *Petrus Abaelardus. Dialectica. First Complete Edition of the Parisian Manuscript with an Introduction*, Assen: Van Gorcum, 1970. Originally written in 1160.

[3] J. Buridan. Summulae de Dialectica. In G. Klima, trans., *Jean Buridan, Summulae de Dialectica, An Annotated Translation with a Philosophical Introduction*, Yale University Press, New Haven, 2001. Originally written in 1487.

[4] R. Descartes. *The Geometry of Rene Descartes*, D. E. Smith and M. L. Lantham, trans., Dover, 1954.

[5] A. Arnauld and P. Nicole. Logic or the Art of Thinking. In T.S. Baynes, trans., J. Gordon, ed., *The Port-Royal Logic, Translated from the French; with Introductions, Notes, and Appendix*, Hamilton, Adams, and Co., London, 1861. Originally written in 1662.

[6] G. Boole. *The Mathematical Analysis of Logic: Being an Essay Towards a Calculus of Deductive Reasoning*, 1847.

[7] A. De Morgan. *Formal Logic: or, The Calculus of Inference, Necessary and Probable*, 1847.

[8] G. Boole. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*, 1854.

[9] T.S. Baynes. *The Port-Royal Logic, Translated from the French; with Introductions, Notes, and Appendix*, J. Gordon, ed., Hamilton, Adams, and Co., London, 1861.

[10] C.S. Peirce. Harvard Lecture 1. In Houser N., et.al., eds., *Writings of Charles S. Peirce*, volume 1, pp. 162-175, 2000. Originally written in 1865.

[11] C.S. Peirce. *Description of a Notation for the Logic of Relatives, Resulting from an Amplification of the Conceptions of Boole's Calculus of Logic*, Welch, Bigelow, and Company, Cambridge, 1870.

[12] G. Frege. Begriffsschrift, A Formal Language, Modeled upon that of Arithmetic, for Pure Thought. In J. van Heijenoort, ed., *From Frege To Godel*, Harvard University Press, Cambridge Massachusetts, 1967. Originally written in 1879.

[13] A. Marquand. A Machine for Producing Syllogistic Variations. In *Studies in Logic: By Members of the Johns Hopkins University*, 1883.

[14] D. Hilbert. Mathematische Probleme. In *Vortrag, gehalten auf dem internationalen Mathematiker-Kongress zu Paris 1900*, Archiv der Mathematik und Physik, 3rd series, 1, 44-63, 213-237, 1900.

[15] L. Couturat. *The Logic of Leibniz in Accordance with Unpublished Documents*, D. Rutherford and R. Timothy Monroe, trans., in progress.

[16] A. Whitehead and B. Russell. *Principia Mathematica*, Cambridge University Press, (vol. 1) 1910, (vol. 2) 1912, (vol. 3) 1913.

[17] D. Hilbert. Neubegründung der Mathematik (Erste Mitteilung). In *Abhandlungen aus dem mathematischen Seminar der Hamburgischen Universität 1*, pp. 157-177, 1922.

[18] P. Bernays and M. Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. In *Mathematische Annelen*, volume 99, pp. 342-372, 1928.

[19] K. Gödel. Die ollständigkeit der Axiome des logischen Funktionenkalküls. In *Monatshefte für Mathematik und Physik 37*, pp. 349, 360.

[20] K. Gödel. Über formal unentscheidbare Sätze der Principia mathematica und verwandter System I. In *Monatshefte für Mathematik und Physik 38*, pp. 173-198, 1931.

[21] J. Herbrand. *Recherches sur la théorie de la démonstation*, Thesis at the University of Paris, 1930.

[22] A. Church. An Unsolvable Problem of Elementary Number Theory. In *American Journal of Mathematics*, volume 58, pp. 345-363, 1936.

[23] A. Church. A Note on the Entscheidungsproblem. In *The Journal of Symbolic Logic*, volume 1, number 1, 1936.

[24] A. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, series 2, number 42, pp. 230-265, 1937.

[25] E. W. Beth. 'Semantic Entailment and Formal Derivability', Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen. In *Afdeling Letterkunde*, N. R. volume 18, number 13, pp. 309-342, 1955.

[26] K. J. J. Hintikka. Form and Content in Quantification Theory. In *Acta Philosohica Fennica*, volume 8, pp. 7-55, 1955.

[27] A. Newell and H.A. Simon. The Logic Theory Machine - A Complex Information Processing System. In *Transactions on Information Theory, IRE.*, volume 2, issue 3, pp. 61-79, 1956.

[28] D. Prawitz, H. Prawitz and N. Voghera. A Mechanical Proof Procedure and its Realization in an Electric Computer. In *Journal of the ACM*, volume 7, issue 2, 1960.

[29] M. Davis and H. Putnam. A Computing Procedure For Quantification Theory. In *Journal of the ACM*, volume 7, issue 3, 1960. Reprinted in [Siekmann, Wrightson 1983].

[30] D. Prawitz. An Improved Proof Procedure. In *Theoria*, volume 26, issue 2, 1960. Reprinted in [Siekmann, Wrightson 1983].

[31] P.C. Gilmore. A Proof Method for Quantification Theory: Its Justification and Realization In *IBM Journal of Research and Development*, volume 4, issue 1, pp. 28-35, 1960. Reprinted in [Siekmann, Wrightson 1983].

[32] H. Wang. Toward Mechanical Mathematics. In *IBM Journal of Research and Development*, volume 4, pp. 2-22, 1960.

[33] W. Kneale and M. Kneale. *The Development of Logic*, Oxford University Press, 1962.

[34] M. Davis, G. Logemann and D. Loveland. A Machine Program for Theorem-proving. In *Communications of the ACM*, volume 5, issue 7, 1962. Reprinted in [Siekmann, Wrightson 1983].

[35] M. Davis. Eliminating the Irrelevant from Mechanical Proofs. In *Proceedings of Symposia in Applied Math*, volume 15, pp. 15-30, 1963. Reprinted in [Siekmann, Wrightson 1983].

[36] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. In *Journal of the Association for Computing Machinery*, volume 12, number 1, pp. 23-41, 1965. Reprinted in [Siekmann, Wrightson 1983].

[37] J. van Heijenoort. *From Frege to Godel*, Harvard University Press, Cambridge Massachusetts, 1967.

[38] J. R. Slagle. Automatic Theorem Proving with Renamable and Semantic Resolution. In *Journal of the ACM*, volume 14, number 4, pp. 687-697, 1967.

[39] D. Loveland. Mechanical Theorem-Proving by Model Elimination. In *Journal of the ACM*, volume 15, issue 2, 1968.

[40] D. Hilbert and P. Bernays. *Foundations Of Mathematics I*, Springer-Verlag, Berlin, 1968.

[41] G. Robinson and L. Wos. Paramodulation and Theorem Proving in First-order Theories with Equality. In *Machine Intelligence 4*, Edinburgh University Press, pp. 135-150, 1969.

[42] R. Kowalski and P.J. Hayes. Semantic Trees in Automatic Theorem Proving. In *Machine Intelligence 4*, Edinburgh University Press, pp. 87-101.

[43] J.L. Bell and A.B. Slomson. *Models and Ultraproducts, An Introduction*, Dover, 1969.

[44] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In *Computational Problems in Abstract Algebra*, Pergamon Press, pp. 263-297, 1970.

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

[46] W. Bibel and J. Schreiber. Proof Search in a Gentzen-like system of first order logic. In *Proceedings of the International Computing Symposium*, North Holland, pp. 205-212, 1975.

[47] D. Brand. Proving Theorems with the Modification Method. In *SIAM Journal on Computing*, volume 4, number 4, pp. 412-430, 1975.

[48] D. W. Loveland. Theorem Proving: a Logical Basis. In *Fundamental Studies in Computer Science*, Elsevier North-Holland, 1978.

[49] G. Nelson and D. Oppen. Simplification by Cooperating Decision Procedures. In *ACM Transactions on Programming Languages and Systems*, volume 1, number 2, pp. 245-257, 1979.

[50] P.B. Andrews. Theorem Proving via General Matings. In *Journal of the Association for Computing Machinery*, volume 28, number 2, pp. 193-214, 1981.

[51] H. Putnam. Pierce the Logician. In *Historia Mathematica*, volume 9, issue 3, pp. 290-301, 1982.

[52] J. Siekmann and G. Wrightson. *Automation of Reasoning. Classical Papers in Computational Logic*, Springer, 1983.

[53] M. Davis. The Prehistory and Early History of Automated Deduction. In *Automation of Reasoning. Classical Papers in Computational Logic*, Springer, 1983.

[54] G. Peterson. A Technique for Establishing Completeness Results in Theorem Proving with Equality. In *SIAM Journal of Computing*, volume 12, issue 1, pp. 82-100, 1983.

[55] R. Shostak. Deciding Combinations of Theories. In *Journal of the ACM*, volume 31, issue 1, pp. 1-12, 1984.

[56] M. Stickel. Automated Deduction by Theory Resolution. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers Inc., pp. 1181-1186, 1985.

[57] P. King. *Jean Buridan's Logic: The Treatise on Supposition; The Treatise on Consequences: Translation from the Latin with a Philosophical Introduction*, Reidel, 1985.

[58] L. Fribourg. A Superposition Oriented Theorem Prover. In *Theoretical Computer Science*, Elsevier Science B.V., volume 35, pp. 129-164, 1985.

[59] D.W. Loveland. Automated Theorem Proving: Mapping Logic into AI. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, Press, pp. 214-229, 1986.

[60] M. Rusinowitch and J. Hsiang. On Word Problems in Equational Theories. In *Automata, Languages and Programming: Lecture Notes in Computer Science*, volume 267, pp. 54-71, 1987.

[61] H. Zhang. *Reduction, superposition and induction: Automated reasoning in an equational logic*, Doctoral Dissertation in Philosophy, Rensselaer Polytechnic Institute, 1988.

[62] H. Zhang and D. Kapur. First-order Theorem Proving Using Conditional Rewrite Rules. In *Proceedings of the 9th International Conference on Automated Deduction*, Lecture Notes in Computer Science, volume 310, pp. 1-20, 1988.

[63] R. Jeroslow. Computation-oriented Reductions of Predicate to Propositional Logic. In *Decision Support Systems*, Elsevier Science B.V., volume 4, issue 2, pp. 183-197, 1988.

[64] L. Bachmair and H. Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In *Proceedings of the 10th International Conference on Automated Deduction*, Springer-Verlag, pp. 427-441, 1990.

[65] S. Lee. *CLIN: An Automated Reasoning System Using Clause Linking*, Doctoral Dissertation in Philosophy, University of North Carolina at Chapel Hill, 1990.

[66] R. Dipert. The Life and Work of Ernst Schroder. In *Modern Logic*, volume 1, pp. 117-139, 1990.

[67] J. Pais and G. Peterson. Using Forcing to Prove Completeness of Resolution and Paramodulation. In *Journal of Symbolic Computation*, volume 11, pp. 3-19, 1991.

[68] J. Goubault. The Complexity of Resource-Bounded First-Order Classical Logic. In *Lecture Notes In Computer Science, Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, volume 775, Springer-Verlag, pp. 59-70, 1994.

[69] S. Lee and D. Plaisted. Problem Solving by Searching for Models with a Theorem Prover. In *Artificial Intelligence*, volume 69, pp. 205-233, 1994.

[70] S. Buss. On Herbrand's Theorem. In *Logic And Computational Complexity*, Lecture Notes In Computer Science, volume 960, Springer-Verlag, pp. 195-209, 1995.

[71] P. King and S. Shapiro. The History Of Logic. In *The Oxford Companion to Philosophy*, Oxford University Press, pp. 496, 1995.

[72] L. Bachmair, H. Ganzinger, C. Lynch and W. Snyder. Basic Paramodulation. In *Information and Computation*, volume 121, number 2, pp. 172-192, 1995.

[73] M. Moser, C. Lynch and C. Steinbach. Model Elimination with Basic Ordered Paramodulation. unpublished, 1995.

[74] C. Barrett, D. Dill and J. Levitt. Validity Checking for Combinations of Theories with Equality. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, Springer-Verlag, pp. 187-201, 1996.

[75] D. Cyrluk, P. Lincoln and N. Shankar. On Shostak's Decision Procedure for Combinations of Theories. In *Proceedings of the 13th International Conference on Automated Deduction*, Springer-Verlag, pp. 463-477, 1996.

[76] J. Billon. The Disconnection Method. In *Proceedings of the 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Springer-Verlag, pp. 110-126, 1996.

[77] N. Bjorner, M. Stickel and T. Uribe. A Practical Integration of First-order Reasoning and Decision Procedures. In *Proceedings of the 14th International Conference on*

Automated Deduction, Lecture Notes in Computer Science, Springer, volume 1249, pp. 101-115, 1997.

[78] A. Leitsch. *The Resolution Calculus, Springer, 1997*

[79] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. In *Journal of Automated Reasoning*, volume 21, number 2, pp. 177-203, 1998.

[80] D. Plaisted and Y. Zhu. Ordered Semantic Hyper-Linking. In *Jornal of Automated Reasoning*, volume 25, number 3, 2000.

[81] A. Tiwari, L. Bachmair and H. Ruess. Rigid E-Unification Revisited. In *Proceedings of the 17th International Conference on Automated Deduction*, Springer-Verlag, pp. 220-234, 2000.

[82] P. Baumgartner. FDPLL - A First Order Davis-Putnam-Logemann-Loveland Procedure. In *Proceedings of the 17th International Conference on Automated Deduction*, Springer-Verlag, pp. 200-219, 2000.

[83] P. Barrouillet. Conditional reasoning by mental models: chronometric and developmental evidence. In *Cognit*, volume 75, pp. 237-266, 2000.

[84] R. Letz and G. Stenz. Proof and Model Generation with Disconnection Tableaux. In *Proceedings of the Conference on Artificial Intelligence on Logic for Programming*, Springer-Verlag, pp. 142-156, 2001.

[85] D. Plaisted. Theorem Proving. In *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, Inc., 2001.

[86] M. Davis. The Early History of Automated Deduction. In *Handbook Of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 3, 2001.

[87] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In *Handbook Of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 19, 2001.

[88] R. Hähnle. Tableaux and Related Methods. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 101, 2001.

[89] M. Baaz, U. Egly and A. Leitsch. Normal Form Transformations. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 273, 2001.

[90] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 371, 2001.

[91] A. Degtyarev and A. Voronkov. Equality Reasoning in Sequent-Based Calculi. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 1, MIT Press, Cambridge, pp. 611, 2001.

[92] C. Fermüller, A. Leitsch, U. Hustadt and T Tammet. Resolution Decision Procedures. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 2, MIT Press, Cambridge, pp. 1791, 2001.

[93] R. Letz and G. Stenz. Model Elimination and Connection Tableau Procedures. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, eds., volume 2, MIT Press, Cambridge, pp. 2015, 2001.

[94] J. Ferreiros. The Road To Modern Logic - An Interpretation. In *The Bulletin of Symbolic Logic*, volume 7, number 4, 2001.

[95] G. Klima. *John Buridan, Summulae de Dialectica, An Annotated Translation, with a Philosophical Introduction*, Yale University Press, New Haven, 2001.

[96] P. King. *The Metaphysics and Natural Philosophy of Buridan*, J.M.M.H Thijssen and J. Zupko, eds., Brill, 2001.

[97] C. Barrett, D.L. Dill and A. Stump. Checking Satisfiability of First-order Formulas by Incremental Translation to SAT. In *Proceedings of the 14th International Conference on Computer Aided Verification*, Springer-Verlag, pp. 236-249, 2002.

[98] C. Tinelli. A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In *Proceedings of the 8th European Confernce on Logics in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Springer, volume 2424, pp. 308-319, 2002.

[99] J.N. Hooker, G. Rago, V. Chandru and A. Shrivastava. Partial Instantiation Methods for Inference in First-order Logic. In *Journal of Automated Reasoning*, volume 28, p. 200, 2002.

[100] S. Shankar. Little Engines of Proof. In *Proceedings of FME 2002, International Symposium of Formal Methods*, Lecture Notes in Computer Science, Springer, volume 2391, pp. 1-20, 2002.

[101] S. Schulz. E - A Brainiac Theorem Prover. In *AI Communications*, IOS Press, volume 15, numbers 2-3/2002, pp. 111-126, 2002.

[102] A. Riazanov and A. Voronkov. The Design and Implementation of VAMPIRE. In *AI Communications*, IOS Press, volume 15, issue 2-3, pp. 91-110, 2002.

[103] A. Riazanov and A. Voronkov. Efficient Instance Retrieval with Standard and Relatinal Path Indexing. In *Proceedings of the 19th International Conference On Automated Deduction*, Lecture Notes in Computer Science, Springer, volume 2741, pp. 380-396, 2003.

[104] P. Baumgartner and C. Tinelli. The Model Evolution Calculus. In *Proceedings of the 19th International Conference On Automated Deduction*, Lecture Notes in Computer Science, Springer, volume 2741, pp. 350-364, 2003.

[105] A. Riazanov. *Implementing An Efficient Theorem Prover*, Doctoral Dissertation in Philosophy, University of Manchester, 2003.

[106] H. Ganzinger and K. Korovin. New Directions in Instantiation-based Theorem Proving. In *Proceedings of 18th IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, pp. 55-64, 2003.

[107] C. Barrett. *Checking Validity of Quantifier-free Formulas in Combination of First-order Theories*, Doctoral Dissertation in Philosophy, Standford University, 2003.

[108] N. Een and N. Sorensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiabilty Testing*, Lecture Notes in Computer Science, Springer, volume 2919, pp. 333-336, 2004.

[109] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras and C. Tinelli. DPLL(T): Fast Decision Procedures. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science, Springer, volume 3452, pp. 175-188, 2004.

[110] J. Marcinkowski, J. Otop and G. Stelmaszek. On a Semantic Subsumtion Test. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science, Springer, volume 3452, pp. 142-153, 2004.

[111] H. Ganzinger and K. Korovin. Integrating Equational Reasoning into Instantiation-based Theorem Proving. In *Proceedings of the 13th Annual Conference of European Association for Computer Science Logic*, Lecture Notes in Computer Science, Springer, volume 3210, pp. 71-84, 2004.

[112] S. Schulz. System Description: E 0.81. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, volume 3097, pages 223-228, 2004.

[113] R. Nieuwenhuis and A. Oliveras. Proof-producing Congruence Closure. In *Proceedings of the 16th International Conference on Term Rewriting and Applications*, Lecture Notes in Computer Science, Springer, volume 3467, pp. 453-468, 2005.

[114] P. Baumgartner and C. Tinelli. The Model Evolution Calculus with Equality. In *Proceedings of the 20th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Springer, volume 3632, pp. 392-408, 2005.

[115] R. Nieuwenhuis, A. Oliveras and C. Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Lecture Notes in Computer Science, Springer, volume 3452, pp. 36-50, 2005.

[116] R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In *Proceedings of the 17th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Springer, volume 3576, pp. 321-334, 2005.

[117] H. Ganzinger and K. Korovin. Theory Instantiation. In *Proceedings of the 13th Conference on Logic for Programming, Artificial, Intelligence and Reasoning*, Lecture Notes in Computer Science, Springer, volume 4246, pp. 497-5111, 2006.

[118] C. Barrett, R. Nieuwenhuis, A. Oliveras and C. Tinelli. Splitting on Demand. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, Springer, volume 4246, pp. 512-526, 2006.

[119] R. Nieuwenhuis, A. Oliveras and C. Tinelli. Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). In *Journal of the ACM*, volume 53, number 6, pp. 937-977, 2006.

[120] B. Lochner. Things to Know when Implementing KBO. In *Journal of Automated Reasoning*, Springer, volume 36, pp. 289-310, 2006.

[121] S. Prestwich and I. Lynce I. Local Search for Unsatisfiability. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, Springer, Aug. 2006 , pp. 283-296, 2006.

[122] S. Delaune, H. Lin and C. Lynch. Protocol Verification Via Rigid/Flexible Resolution. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pp. 242-256, 2007.

[123] Y. Ge, C. Barrett and C. Tinelli. Solving Quantified Verification Conditions Using Satisfiability Modulo Theories. In *Proceedings of the 21st International Conference on Automated Deduction*, Lecture Notes in Computer Science, Springe-http://www.cs.nyu.edu/acsys/cvc3/doc/index.htmlr, volume 4603, pp. 167-182, 2007.

[124] C. Barrett and C. Tinelli. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification*, Springer-Verlag, volume 4590, pp. 298-302, 2007.

[125] P. Baumgartner. Logical Engineering with Instance-based Methods. In *Proceedings of the 21st International Conference on Automated Deduction*, Springer-Verlag, pp. 404-409, 2007.

[126] T. Deshane, W. Hu, P. Jablonski, H. Lin, C. Lynch and R.E. McGregor. Encoding First Order Proofs in SAT. In *Proceedings of the 21st International Conference on Automated Deduction*, Springer-Verlag, pp. 476-491, 2007.

[127] M. Ludwig and U. Waldmann. An extension of the Knuth-Bendix Ordering with LPO-like properties. In *Proceedings of the 14th international conference on Logic for Programming*, Springer-Verlag, pp. 348-362, 2007.

[128] J. Bonjio, C. Katrak, H. Lin, C. Lynch and R.E. McGregor. Encoding First Order Proofs in SMT. In *Electronic Notes in Theoretical Computer Science*, volume 198, number 2, pp. 71-84.

[129] P. Baumgartner and C. Tinelli. The Model Evolution Calculus As A First-order DPLL Procedure. In *Artificial Intelligence*, Elsevier Science Publishers Ltd., volume 172, issue 4-5, pp. 591-632, 2008.

[130] L. de Moura and N. Bjorner. Engineering DPLL(T) + Saturation. In *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Springer-Verlag, pp. 475-490, 2008.

[131] G. Sutcliffe. The CADE-21 Automated Theorem Proving System Competition. In *AI Communications*, volume 21, number 1, pp. 71-82.

[132] C. Lynch and D. Tran. SMELS: Satisfiability Modulo Equality with Lazy Superposition. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, Springer, volume 5311, pp. 186-200, 2008.

[133] K. Korovin. An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Springer-Verlag, pp. 292-298, 2008.

[134] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, Prentice Hall, 3rd Edition, 2009.

[135] C. Sticksel. Efficient Ground Satisfiability Solving In An Instantiation-based Method For First-order Theorem Proving. Presented at *the 16th Workshop on Automated Reasoning*, 2009.

[136] C. Lynch and R.E. McGregor. Combining Instance Generation and Resolution. In *Proceedings of the 7th International Conference on Frontiers of Combining Systems*, Springer-Verlag, pp. 304-318, 2009.

[137] S. Jabobs. Incremental Instance Generation in Local Reasoning. In *Proceedings of the 21st International Conference on Computer Aided Verification*, Springer-Verlag, pp. 368-382, 2009.

[138] L. de Moura and N. Bjorner. Satisfiabilty Modulo Theories: An Appetizer. In *Formal Methods: Foundations and Applications*, volume 5902, pp. 23-36, 2009.

[139] Y. Ge and L. de Moura. Complete Instantiation for Quantified SMT Formulas. In *Proceedings of the 21st International Conference on Computer Aided Verification*, Springer-Verlag, pp. 306-320, 2009.

[140] P. Baumgartner and U. Waldmann. Superposition and Model Evolution Combined. In *Proceedings of the 22nd International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, volume 5663, pp. 17-34, 2009.

[141] M.P. Bonacina, C. Lynch and L. de Moura. On Deciding Satisfiability by DPLL(Gamma+T) and Unsound Theorem Proving. In *Proceedings of the 22nd International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, volume 5663, pp. 35-50, 2009.

[142] K. Korovin and C. Sticksel. iProver-Eq: An Instantiation-based Theorem Prover with Equality. In *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Springer, pp. 196-202, 2010.

[143] P. Baumgartner and E. Thorstensen. Instance Based Methods - An Overview. In *Kunstliche Intelligenz*, volume 24, number 1, pp. 35-42, 2010.

[144] M. Paola-Bonacina, C.A. Lynch and L. de Moura. On Deciding Satisfiability by Theorem Proving with Speculative Inference. In *Journal of Automated Reasoning*, Springer Netherlands, pp. 1-29, 2010.

[145] S. Schulz. E 1.2 User Manual. `http://www4.infomatik.tu-meunchen.de/~schulz/WORK/eprover.ps`.

[146] C. Barrett. CVC3 Documentation. `http://www.cs.nyu.edu/acsys/cvc3/doc/index.html`.

[147] B. Dutertre and L. de Moura. Yices. `http://yices.csl.sri.com`.