Combining Instance Generation and Resolution

Christopher Lynch and Ralph Eric McGregor

Clarkson University
www.clarkson.edu/projects/carl

Abstract. We present a new inference system for first-order logic, SIG-Res, which couples together SInst-Gen and ordered resolution into a single inference system. Given a set F of first order clauses we create two sets, P and R, each a subset of F. Under SIG-Res, P is saturated by SInst-Gen and resolution is applied to pairs of clauses in $P \cup R$ where at least one of the clauses is in R. We discuss the motivation for this inference system and prove its completeness. We also discuss our implementation called Spectrum and give some initial results.

1 Introduction

The most efficient propositional logic SAT solvers are those based on the DPLL procedure [4, 5]. Given the incredible efficiency of these solvers, a great deal of research has centered around ways to utilize the efficiency of SAT solvers for first-order logic theorem proving. Toward this pursuit, some have lifted the DPLL procedure to first-order logic in the form of the model evolution calculus [3]. Others have used SAT solvers as auxiliary tools to determine the satisfiability of certain fragments (or all) of first-order logic via propositional encodings [7, 11, 22]. A third use of SAT solvers in first-order logic theorem proving has been in saturation-based instance generation methods which repeatedly call upon a SAT solver to find so-called *conflicts* between clauses and use instance generation inferences to resolve these conflicts. Notable in this last line of research are the works by Jeroslow [12, 13] who developed the saturation-based instance generation method called *Partial Instantiation* (PI), Hooker (et.al) who developed the first complete PI method for the full first-order logic called *Primal PI* [10] and Ganzinger and Korovin who among many other contributions formalized and proved the completeness of the instance generation inference rule, Inst-Gen, and the extention SInst-Gen (Inst-Gen with semantic selection and hyper inferences) in [9]. The efficiency of saturation-based instance generation methods is demonstrated by Korovin's implementation called IProver [15, 24].

Another well known line of research in first-order logic theorem proving began with Robinson's landmark paper [20] which describes his resolution principle and unification. Since then, many refinements have been made, e.g. ordered resolution and semantic resolution [1, 16, 18]. Recent implementations of resolution have been shown to be the top performer in the CASC competition [24].

A key benefit of resolution is the ability to restrict the search space with the use of *ordered* resolution. Here, resolution inferences are only necessary on

maximal literals. Selection rules can also be applied which also restrict the search space. 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 inference among two ground clauses will produce another ground clause which does not introduce any new literals. Any 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. However, 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.

Although resolution methods appear to be more efficient in practice, there are some classes of problems that are suited better for instantiation-based methods. 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. There is even a class of first-order logic problems, problems which contain no function symbols, called Effectively Propositional (EPR) for which instantiation-based methods form a decision procedure, whereas resolution methods may run forever.

Here 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. The inference system SIG-Res, given in this paper, 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 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 [17] and the DPLL($\Gamma + T$) inference system [19]. 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.

In the pages that follow, we prove the completeness of our inference system, discuss our implementation, called Spectrum, and we present our initial implementation results.

2 Preliminaries

We begin by defining the terminology used in this paper as is in [1,9]. The setting we are considering is classical first order logic. Variables and constant symbols are *terms* and 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. If P is a predicate symbol and $t_1, \ldots, t_n$ are terms then $P(t_1, \ldots, t_n)$ is an *atom*. Formulas are constructed using the logical connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) under the standard rules of formula construction. A *literal* is either an atom (positive literal) or the negation of an atom (negative literal). If A is an atom we say that A and $\neg A$ are *complements* and we denote the complement of a literal L by $\overline{L}$. 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 $\Box$.

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 $A < \neg A$. 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 $\max(C) = \{L | L \text{ is maximal in } C\}$.

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 literals L and K is a substitution σ such that $L\sigma = K\sigma$. If such a unifier exists, we say that L and K are *unifiable*. A *most general unifier* of L and K, denoted $\text{mgu}(L, K)$, 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 $Gr(S)$.

$\perp$ is used to denote a distinguished 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 $\bot$-substitution to the clauses in P.

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.

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.

3 Instance Generation and 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 [9] inference rule. An instance generation with semantic selection inference system (SInst-Gen) (See Figure 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, $\text{sel}(C, I_\bot)$, 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 $\Gamma \vee L$ and $\Delta \vee \overline{K}$ *conflict* if

(i) $L \in sel(\Gamma \vee L, I_\bot)$ and $\overline{K} \in \text{sel}(\Delta \vee \overline{K}, I_\bot)$
(ii) L and K are unifiable

Instance generation with semantic selection methods saturate a set of clauses P by repeatedly calling apon a SAT solver to obtain a model for $P\bot$ and resolving all conflicts with SInst-Gen inferences [1]. 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 [9] but may not halt.

The ordered resolution and factoring inference rules are well known in the literature. For completeness they are given in Figure 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.

[1] Instance generation with selection functions mapping to singleton sets and resolving conflicts based on these selected literals is precisely the Primal PI method in [10]

SInst-Gen

$$\frac{\Gamma \vee L \qquad \Delta \vee \overline{K}}{(\Gamma \vee L)\tau \qquad (\Delta \vee \overline{K})\tau}$$

where (i) $L \in sel(\Gamma \vee L, I_\perp)$ and $\overline{K} \in \text{sel}(\Delta \vee \overline{K}, I_\perp)$
(ii) $\tau = \text{mgu}(L, \overline{K})$

Fig. 1. SInst-Gen Inference Rule

The satisfiablity 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 ($\square$) is resolved, in which case R is unsatisfiable, or the set is saturated and $\square \notin R$, 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.

Ordered Resolution

$$\frac{\Gamma \vee L \qquad \Delta \vee \overline{K}}{(\Gamma \vee \Delta)\tau}$$

where (i) $L \in \max(\Gamma \vee L)$ and $\overline{K} \in \max(\Delta \vee \overline{K})$
(ii) $\tau = \text{mgu}(L, K)$

Factoring

$$\frac{\Gamma \vee L \vee K}{(\Gamma \vee L)\tau}$$

where $\tau = \text{mgu}(L, K)$

Fig. 2. Ordered Resolution and Factoring Inference Rules

4 SIG-Res

The inferences in SIG-Res are variations of SInst-Gen, ordered resolution and factoring (see Figure 3). SIG-Res is an inference system that requires two sets of clauses. Given a set of clauses, 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 6 we describe one general heuristic, GSM, which we have incorporated into our implementation.

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

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

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 implmented IG-Res in a theorem prover called Spectrum. Our implementation uses the latter method and follows Algorithm 1.

5 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

SInst-Gen

$$\frac{\Gamma \vee L \qquad \Delta \vee \overline{K}}{(\Gamma \vee L)\tau \qquad (\Delta \vee \overline{K})\tau}$$

where (i) $\Gamma \vee L \in P$ and $\Delta \vee \overline{K} \in P$
(ii) $L \in \mathrm{sel}(\Gamma \vee L, I_\perp)$ and $\overline{K} \in \mathrm{sel}(\Delta \vee \overline{K}, I_\perp)$
(iii) $\tau = \mathrm{mgu}(L, K)$
(iv) $(\Gamma \vee L)\tau \in P$ and $(\Delta \vee \overline{K})\tau \in P$

Ordered Resolution

$$\frac{\Gamma \vee L \qquad \Delta \vee \overline{K}}{(\Gamma \vee \Delta)\tau}$$

where (i) $\Gamma \vee L \in R$ or $\Delta \vee \overline{K} \in R$
(ii) $L \in \mathrm{sel}(\Gamma \vee L, I_\perp)$ and $\overline{K} \in \mathrm{sel}(\Delta \vee \overline{K}, I_\perp)$
(iii) $\tau = \mathrm{mgu}(L, K)$
(iv) $(\Gamma \vee \Delta)\tau \in P$ if $\Gamma \vee L \notin R$ or $\Delta \vee \overline{K} \notin R$

Factoring

$$\frac{\Gamma \vee L \vee K}{(\Gamma \vee L)\tau}$$

where (i) $\tau = \mathrm{mgu}(L, K)$
(ii) $(\Gamma \vee L)\tau \in P$ if $\Gamma \vee L \vee K \notin R$

Fig. 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 1 *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 [9], 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 2 *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 $D = D' \cdot \sigma$ [2] 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 $\tau = \text{mgu}(\overline{K}, L)$ 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.

[2] 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.

6 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 [25] and outputs results in accordance to the SZS ontology [23]. 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 1.

As we begin the instance generation phase on the set P, Yices [8] 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.

7 Experimental Results and Example

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* threorem 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 [3]. 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,

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

Algorithm 1 Spectrum(P,R)

```
while true do
    N_P := ∅
    N_R := ∅

    Run SAT on P⊥
    if P⊥ is unsatisfiable then
        return UNSATISFIABLE
    for all C_1, C_2 ∈ P do
        if conflict(C_1, C_2) = true then
            N_P := N_P ∪ (SInst-Gen(C_1, C_2) \ P)

    for all C_1 ∈ P, C_2 ∈ R do
        D := Resolution(C_1, C_2)
        if □ ∈ D then
            return UNSATISFIABLE
        else if D ≠ ∅ then
            N_P := N_P ∪ (D \ P)

    for all C_1, C_2 ∈ R do
        D := Resolution(C_1, C_2)
        if □ ∈ D then
            return UNSATISFIABLE
        else if D ≠ ∅ then
            for all C ∈ D do
                F := Factor(C)
                for all B ∈ F do
                    T := distribute(B)
                    N_T := N_T ∪ ({B} \ T)

    if N_P = ∅ and N_R = ∅ then
        return SATISFIABLE
    else
        Set P = P ∪ N_P
        Set R = R ∪ N_R
```

since they have only a single maximal literal, are put in R which avoids this problem.

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. 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.

8 Conclusion

In this paper, we have developed an inference system which combines instance generation and resolution. We have proved its completeness and provide some preliminary experimental results from our implementation. The key feature of our inference system is that clauses are partitioned into two sets: P and R. SInst-Gen inferences are performed between members of P, while resolution is performed between clauses when one is from R.

It is important to provide a good heuristic to decide which clauses should be in P and which ones should be in R. Ground clauses ought to be in P, because the SAT solver processes ground clauses most efficiently. Our heuristic puts clauses with more than one maximal literal into P, because ordered resolution generally does not handle these clauses well. For example, ordered resolution with the Transitivity Axiom does not halt. In our heuristic, we put clauses with a single maximal literal into R, because ordered resolution with those literals will generally reduce the size of the other premise.

Our implementation is rudimentary and does not contain all the useful features of state of the art theorem provers but is still useful for comparison purposes. There are several examples from LCL, and also the GRP problem we illustrate, 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 instance generation. While at the same time, it contains clauses similar to Transitivity where ordered resolution

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. $P(id, x, x)$

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

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

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

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

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

Fig. 4. Proof of GRP006-1

is explosive. We believe these examples show the use of our technique and the potential for further research into this area.

9 Future Work

We are continuing our experiments to determine which distribution heuristics perform best on general sets of problems and for certain classes of problems. We are also continuing the development of Spectrum. As a rudimentary theorem prover, there is room for improving Spectrum's performance by incorporating more sophisticated data structures, heuristics and techniques that are in the literature, e.g. implementing a more sophisticated method for choosing selected literals for clauses in P, restricting SInst-Gen inference using *dismatching constraints*, using more efficient factoring and redundancy elimination techniques, etc.

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.

We are also interested in investigating if the partitioning idea can be extended to equalities. Specifically, given a problem, we are interested in developing a method which uses a SMT solver to solve the ground equalities and Rewriting techniques to solve the non-ground equalities.

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?

References

1. L. Bachmair, H. Ganzinger. Resolution Theorem Proving. In J. A. Robinson and A. Voronkovs, editors, Handbook of Automated Reasoning, volume 1, chapter 2, pages 19-99. Elsevier and MIT Press, 2001.
2. P. Baumgartner. Logical Engineering With Instance-Based Methods. In F. Pfennings, editor, Proceedings of the Twenty-first International Conference on Automated Deduction (CAV'07), Berlin, Germany, volume 4590 of Lecture Notes in Computer Science, pages 298-302. Springer 2007.
3. P. Baumgartner and C. Tinelli. The Model Evolution Calculus as a First-Order DPLL Method. Artificial Intelligence, 172:591-632, 2008.
4. M. Davis, G. Logemann, D. Loveland. A Machine Program for Theorem Proving. Communications of the ACM, volume 5, issue 7, pages 394-397. ACM, 1962.
5. M. Davis, H. Putnam. A Computing Procedure for Quantification Theory. Journal of the ACM, volume 7, issue 3, pages 201-215. ACM, 1960.
6. L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, Lecture Notes in Computer Science. Springer, 2008.
7. T. Deshane, W. Hu, P. Jablonski, H. Lin, C. Lynch, R. E. McGregor. Encoding First Order Proofs in SAT. In Proceedings of The Conference on Automated Deduction,

(CADE'07), volume 4603 of Lecture Notes in Computer Science, pages 476-491. Springer, 2007.
8. B. Dutertre, L. de Moura. The Yices SMT Solver. Available at http://yices.csl.sri.com/tool-paper.pdf
9. H. Ganzinger and K. Korovin. New Directions in Instantiation-Based Theorem Proving. In Proc. 18th IEEE Symposium on Logic in Computer Science, (LICS'03), pages 55-64. IEEE Computer Society Press, 2003.
10. J. N. Hooker, G. Rago, V. Chandru, A. Shrivastava. Partial Instantiation Methods for Inference in First Order Logic. Journal of Automated Reasoning, volume 28, number 4, pages 371-396. Springer Netherlands, 2002.
11. D. Jackson. Automating First-Order Relational Logic. ACM SIGSOFT Software Engineering Notes, volume 25, issue 6, pages 130-139. ACM, 2000.
12. R. G. Jeroslow. Computation-Oriented Reductions of Predicate to Propositional Logic. Decision Support Systems, volume 4, pages 183-197. Elsevier Science, 1988.
13. R. G. Jeroslow. Logic-Based Decision Support: Mixed Integer Model Formulation. Annals of Discrete Mathematics, volume 40. North-Holland, 1989.
14. K. Korovin. An Invitation to Instantiation-Based Reasoning: From Theory to Practice. Volume in Memoriam of Harald Ganzinger. LCNS, to appear.
15. K. Korovin. System Description: iProver - An Instantiation-Based Theorem Prover for First-Order Logic. In Proceedings of the 4th International Joint Conference on Automated Reasoning, (IJCAR'08), volume 5195 of Lecture Notes In Artificial Intelligence, pages 292-298. Springer-Verlag, 2008.
16. A. Leitsch. The Resolution Calculus. Springer 1997.
17. C. Lynch, D. Tran. SMELS: Satisfiability Modulo Equality with Lazy Superposition. 6th International Symposium on Automated Technology for Verification and Analysis, volume 5311 of Lecture Notes in Computer Science, pages 168-200. Springer, 2008.
18. D. W. Loveland. Automated Theorem Proving: a Logical Basis (Fundamental Studies in Computer Science). Elsevier North-Holland 1978.
19. M. Paola Bonacina, C. Lynch and L. de Moura. On Deciding Satisfiability by DPLL(Gamma+T) and Unsound Theorem Proving. To appear in The Proceedings of The 22nd International Conference on Automated Deduction, (CADE'09). Springer, 2009.
20. J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. J. Association for Computing Machinery, volume 12, pages 23-41. ACM, 1965.
21. J. R. Slagle. Automatic Theorem Proving with Renamable and Semantic Resolution. Journal of the ACM, volume 14, issue 4, pages 687-697. ACM, 1967.
22. O. Strichman, S. A. Seshia, R. E. Bryant. Deciding Separation Formulas With SAT. In Proceedings of the Computer Aided Verification Conference, (CAV'02), volume 2404 of Lecture Notes in Computer Science, pages 113-124. Springer 2002.
23. G. Sutcliffe. The SZS Ontology. Available at http://www.cs.miami.edu/ tptp.
24. G. Sutcliffe. The CADE-21 Automated Theorem Proving System Competition. AI Communications, volume 21, number 1, pages 71-82.
25. G. Sutcliffe, C. B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning, volume 21, number 2, pages 177-203. Kluwer Academic Publishers, 1998.