Inst-Gen – A Modular Approach to Instantiation-Based Automated Reasoning

Konstantin Korovin*

The University of Manchester
School of Computer Science
korovin@cs.man.ac.uk

Abstract. Inst-Gen is an instantiation-based reasoning method for first-order logic introduced in [18]. One of the distinctive features of Inst-Gen is a modular combination of first-order reasoning with efficient ground reasoning. Thus, Inst-Gen provides a framework for utilising efficient off-the-shelf propositional SAT and SMT solvers as part of general first-order reasoning. In this paper we present a unified view on the developments of the Inst-Gen method: (i) completeness proofs; (ii) abstract and concrete criteria for redundancy elimination, including dismatching constraints and global subsumption; (iii) implementation details and evaluation.

1 Introduction

The basic idea behind instantiation-based reasoning is to interleave smart generation of instances of first-order formulae with propositional type reasoning. Instantiation-based methods can be divided into two major categories: (i) fine-grain interleaving of instantiation with efficient propositional inference rules, and (ii) modular combination of instantiation and propositional reasoning. One of the most prominent examples from the first category is the model evolution calculus (ME) [8] which interleaves instance generation with DPLL style reasoning. The model evolution calculus is implemented in a reasoning system called Darwin [6].

Our approach to instantiation-based reasoning [18] falls into the second category, where propositional reasoning is integrated in a modular fashion and was inspired by work on hyper-linking and its extensions (see, [24, 35, 43]). The main advantage of the modular combination of propositional reasoning is that it allows one to use off-the-shelf SAT and SMT solvers in the context of first-order reasoning. One of our main goals is to develop a flexible theoretical framework, called Inst-Gen, for modular combination of instantiation with propositional reasoning and more generally with ground reasoning modulo theories. This framework provides methods for proving completeness of instantiation calculi, powerful redundancy elimination criteria and flexible saturation strategies. All these ingredients are crucial for developing reasoning systems which can be used in practical applications. We also show that most of the powerful machinery developed in the resolution-based framework (see [3, 38]) can be suitably adapted for

* Supported by a Royal Society University Research Fellowship

the Inst-Gen method. Based on these theoretical results we have developed and implemented an automated reasoning system, called iProver [31]. iProver features state-of-the-art implementation techniques such as unification and simplification indexes; semantically-guided inferences based on propositional models; redundancy elimination based on dismatching constraints, blocking of non-proper instantiations and global subsumption. For propositional reasoning iProver uses an optimised propositional SAT solver MiniSAT [15].

One of the major success stories of instantiation-based methods is in reasoning with the effectively propositional (EPR) fragment of first-order logic, also called the Bernays-Schönfinkel class. All known instantiation-based methods are decision procedures for the EPR fragment and experimental results show that instantiation-based methods considerably outperform other methods on this fragment. In particular, iProver has been winning the EPR division of the world championship for automated theorem proving (CASC)[1] for the last four years. Recently it was shown that the EPR fragment has a number of applications in areas such as bounded model checking, panning, logic programming and knowledge representation [16, 17, 25, 39, 40, 49]. The importance of the EPR fragment has triggered the development of a number of dedicated methods [7, 12, 41], but these are yet to be extensively evaluated and compared with general-purpose instantiation-based methods.

In this paper we present a unified view on the developments of the Inst-Gen method from theoretical foundations to implementation and evaluation: (i) completeness proofs; (ii) abstract and concrete criteria for redundancy elimination; and (iii) implementation of Inst-Gen in iProver and evaluation.

This paper is structured as follows. Preliminaries are in Section 2. In Section 3 we introduce the Inst-Gen calculus which is the basis of our framework. We show how instantiation process can be guided by propositional models of ground abstractions in Section 4. Simplifications and redundancy elimination which are crucial for practical applicability of the method are described in Sections 5–7. A combination of instantiation with resolution is described in Section 8. We discuss strategies for interleaving application of inference rules, simplifications and propositional reasoning in Section 9. In Section 10 we show that Inst-Gen is a decision procedure for the EPR fragment. In Section 11 we discuss implementation of Inst-Gen in iProver. iProver is evaluated in Section 12.

2 Preliminaries

We adopt standard terminology used in first-order reasoning [3, 38]. Let $\Sigma = \langle \mathcal{P}, \mathcal{F} \rangle$ be a first-order signature, where $\mathcal{P}$ is the set of predicate symbols and $\mathcal{F}$ is the set of function symbols. We assume that $\mathcal{F}$ contains a designated constant $\bot$ (not to be confused with falsum). Let $\mathcal{V}$ be a set of variables. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ will be denoted as $\mathcal{T}(\mathcal{F}, \mathcal{V})$.

A *substitution* is a mapping from variables into terms which is the identity on all but finitely many variables. Substitutions will be denoted by ρ, σ, τ, and θ.

[1] http://www.cs.miami.edu/~tptp/CASC/

A *clause* is a possibly empty multiset of literals denoting their disjunction and is usually written as $L_1 \vee \ldots \vee L_n$, where a *literal* being either an atomic formula or the negation thereof. We say that C is a subclause of D, and write $C \subseteq D$, if C is a submultiset of D. Variables are usually denoted by x, y, and z, whereas, letters a, b and c denote constants. If L is a literal, $\overline{L}$ denotes the complement of L.

A substitution is called a *proper instantiator* of an expression (a literal or clause) if at least one variable of the expression is mapped to a non-variable term, otherwise it is called a *non-proper instantiator*. *Renamings* are injective substitutions, mapping variables to variables. Two clauses are *variants* of each other if one can be obtained from the other by applying a renaming. We will ambiguously use $\bot$ to denote also the substitution mapping all variables to the constant $\bot$. If S is a set of clauses, by $S\bot$ we denote all ground clauses obtained by applying $\bot$ to each clause in S.

We will be working with a refined notion of instances of clauses, called closures. A *closure* is a pair consisting of a clause C and a substitution σ written $C \cdot \sigma$. A closure $C \cdot \sigma$ *represents* the clause $C\sigma$. Let us note that a clause generally has more than one representation by closures. A closure is called *ground* if it represents a ground clause. In this paper we mainly consider ground closures and will implicitly assume that closures are ground unless specified otherwise. We work modulo renaming, that is, we do not distinguish between closures $C \cdot \sigma$ and $D \cdot \tau$ for which C is a variant of D and $C\sigma$ is a variant of $D\tau$. Let S be a set of clauses and C a clause in S, then a ground closure $C \cdot \sigma$ is called a *ground instance* of C in S and we also say that the closure $C \cdot \sigma$ is a *representation* of the clause $C\sigma$ in S.

Our restrictions on the instantiation calculus and completeness proofs are based on an ordering on closures defined as follows. A *closure ordering* is any ordering $\succ$ on closures that is total, well-founded and satisfies the following condition. If $C \cdot \sigma$ and $D \cdot \tau$ are such that $C\sigma = D\tau$ and $C\theta = D$ for some proper instantiator θ, then $C \cdot \sigma \succ D \cdot \tau$. In particular, more specific representations of the same ground clause are smaller in this ordering than more general representations. For example, $(A(x, y) \vee B(y)) \cdot [f(c)/x, c/y] \succ (A(f(u), v) \vee B(v)) \cdot [c/u, c/v]$. It is easy to see that any well-founded ordering on ground clauses can be extended to a total well-founded closure ordering. For the rest of this paper we assume that $\succ$ is a closure ordering.

We consider *Herbrand interpretations* which are sometimes partial, given by consistent sets I of ground literals. A ground literal L is called *undefined* in I if neither L nor $\overline{L}$ is in I. I is called *total* if for each ground literal, I either contains the literal or its complement. A clause C is *true* (or *valid*) in a partial interpretation I, written $I \models C$, if C is true in every total extension of I, and is called *false* (*not valid*) in I, otherwise. We say that an interpretation I is a *model* of a set of clauses S if all clauses in S are true in I. We say that a set of clauses S *entails* a set of clauses S', denoted by $S \models S'$, if every Herbrand model of S is a model of S'. The truth value of a closure is defined to be equal to the truth value of the clause it represents.

3 Instantiation Calculus

The basic idea behind the modular approach to instantiation-based reasoning is to approximate the unsatisfiability problem for sets of first-order clauses by a sequence of

propositional problems. This can be done in the following way. Let S be a set of first-order clauses. We first abstract S by a set of propositional clauses $S\bot$, obtained by mapping all variables into a distinguished constant $\bot$. If the propositional abstraction $S\bot$ is unsatisfiable (which can be shown by any propositional solver), then S is also unsatisfiable and we are done. If $S\bot$ is satisfiable then it still possible that S is unsatisfiable and we need to add more instances of clauses to S witnessing the unsatisfiability of S at the ground level. The process continues by refining the ground abstraction adding appropriate instances of clauses to S, until either an unsatisfiable ground abstraction is obtained or, possibly in the limit, a set of clauses that can not be refined further. In the former case the initial set of first-order clauses is unsatisfiable and in the latter case we can show that the initial set clauses is satisfiable.

There are three major issues to consider:

1. how to generate instances;
2. how to interleave propositional reasoning and instantiation;
3. how to guide instance generation and reduce the number of redundant instances.

For the generation instances we use the Inst-Gen calculus and its refinements described below. In the later sections we address the rest of these issues. Informally, we instantiate clauses in S by applying the Inst-Gen inference rule or its refinements, until we either obtain: i) a set of clauses with unsatisfiable ground abstraction; or ii) a saturated set, i.e. no new instances can be derived by Inst-Gen. In the former case the soundness of the Inst-Gen calculus implies that S is unsatisfiable, in the latter case, completeness of Inst-Gen implies that S is satisfiable (Theorem 1).

Consider a set of clauses S and assume that the ground abstraction $S\bot$ of S is satisfiable.

Inst-Gen

$$\frac{L \vee C \quad \overline{L}' \vee D}{(L \vee C)\theta \;\; (\overline{L}' \vee D)\theta}$$

where (i) θ is the most general unifier (mgu) of L and L',
(ii) θ is a proper instantiator of L or L'.

The Inst-Gen inference rule resembles resolution but instead of resolving we instantiate the premises, leaving to propositional reasoning to deal with the obtained instances.

The soundness of Inst-Gen is obvious: conclusions of Inst-Gen logically follow from the premises. We say that a set of clauses S is *Inst-Gen saturated* if the conclusion of any Inst-Gen inference with premises in S is also in S. In other words, if a set of clauses S is Inst-Gen saturated, then we cannot refine our ground abstraction further using Inst-Gen. Our first completeness Theorem 1 implies that in this case, satisfiability of the ground abstraction $S\bot$ is equivalent to satisfiability of the set of first-order clauses S.

Theorem 1. [18] *Let S be an Inst-Gen saturated set of clauses. Then S is satisfiable if, and only if, $S\bot$ is satisfiable.*

Theorem 1 will follow from a more general Theorem 4 proved later. Let us consider a simple example.

Example 1. Let S be the following set of clauses.

S		$S\bot$
$A(f(x), b) \vee B(x, y)$	(1)	$A(f(\bot), b) \vee B(\bot, \bot)$
$\neg A(f(f(x)), y)$	(2)	$\neg A(f(f(\bot)), \bot)$
$\neg B(f(x), x)$	(3)	$\neg B(f(\bot), \bot)$

First we note that the propositional abstraction $S\bot$ of S is satisfiable. Applying Inst-Gen to (1) and (2) we obtain

$$S_1 = S \cup \{A(f(f(x)), b) \vee B(f(x), y); \neg A(f(f(x)), b)\}.$$

Now it is easy to see that $S_1\bot$ is unsatisfiable and we can use any propositional solver to show this. On the other hand, if we consider a set of clauses S' consisting of (1) and (2), then after applying Inst-Gen we obtain an Inst-Gen saturated set of clauses and Theorem 1 implies that S' is satisfiable.

One can think of Inst-Gen instantiation inferences as refinements of the ground abstraction in the following sense. Consider two closures $C \cdot \sigma$ and $D \cdot \tau$ representing the same ground clause G, i.e., $C\sigma = D\tau = G$. We say that $C\cdot\sigma$ is a finer representation of G than $D \cdot \tau$ if $D \cdot \tau \succ C \cdot \sigma$. In particular, if C is a proper instance of D then all ground closures of C are finer representations than corresponding ground closures of D. A ground abstraction $D\bot$ of a clause $D \in S$ is intended to represent all ground instances of D which do not have a finer representation in S. Such an abstraction may need to be refined if we have a clause $(L \vee C) \in S$ and a literal $\overline{L}' \in D$ such that L and L' share a common instance, i.e unifiable, but L and L' have different abstractions, i.e. $L'\bot \neq L\bot$. Such refinements are reflected by the Inst-Gen inference system.

Inst-Gen is similar to resolution but instead of producing a resolvent we generate corresponding instances of clauses, leaving the option of producing the resolvent to the propositional reasoner. There are also subtle differences between Inst-Gen and resolution. First, resolution usually produce clauses with increasing number of literals. Inst-Gen generates instances of the initial clauses and therefore the number of literals in clauses does not increase. In particular, Inst-Gen is a decision procedure for the effectively propositional fragment – the clausal fragment where the signature is restricted to contain only predicate symbols and constants (see Section 10). Second, there is no recombination of clauses which can result in repeated work in the resolution setting (see [35] and Example 5).

Let us observe that Inst-Gen uses most general unifiers to focus on relevant conflicting instances of clauses. Inst-Gen already features some restrictions on applicability imposed by unification and requiring the unifier to be a proper instantiator; nevertheless Inst-Gen is a very prolific inference system. One source of inefficiency is that any literal in a clause can participate in an Inst-Gen inference. Next, we show how to restrict Inst-Gen to only selected literals based on a semantic criterion.

4 Semantic Selection and Hyper-inferences

Semantic selection is motivated by the following observation. Let S be a set of clauses such that its propositional abstraction $S\bot$ is satisfiable. Let $I_\bot$ be a propositional model of $S\bot$. We try to extend $I_\bot$ to a first-order model of all ground instances of clauses in S by taking the truth value of a literal $L\sigma$ to be the truth value of $L\bot$ in $I_\bot$. Such an extension can rise to conflicts. Let S be the set of clauses $\{A(f(x)) \vee C(x); \neg A(y) \vee D(y)\}$ and $I_\bot$ a model of $S\bot$. A conflict arises if both $A(f(\bot))$ and $\neg A(\bot)$ are true in $I_\bot$, as, e.g., assigning true to both $A(f(a))$ and $\neg A(f(a))$ would result in an inconsistent interpretation. We can resolve this conflict by applying Inst-Gen inference obtaining $S_1 = S \cup \{A(f(x)) \vee C(x); \neg A(f(y)) \vee D(f(y))\}$. Now the propositional solver is supplied with the necessary information about instances of clauses with conflicting literals and a propositional model of $S_1\bot$ can be extended to a first-order model of S_1. This can be generalised to restrict Inst-Gen inferences to resolve only conflicts relevant to a propositional model of the propositional abstraction of the current set of clauses.

A *selection function* sel for a set of clauses S is a mapping from clauses in S to literals such that $\mathsf{sel}(C) \in C$ for each clause $C \in S$. We say that sel is based on a model $I_\bot$ of $S\bot$, if $I_\bot \models \mathsf{sel}(C)\bot$ for all $C \in S$. Let S be a set of clauses such that $S\bot$ is consistent and sel be a selection function based on a model $I_\bot$ of $S\bot$. Then, the instance generation calculus SInst-Gen based on sel, is defined as follows.

SInst-Gen

$$\frac{L \vee C \quad \overline{L}' \vee D}{(L \vee C)\theta \quad (\overline{L}' \vee D)\theta}$$

where (i) θ is the most general unifier of L and L', and
(ii) $\mathsf{sel}(L \vee C) = L$ and $\mathsf{sel}(\overline{L}' \vee D) = \overline{L}'$.

Although we have omitted the requirement on θ to be a proper instantiator, this condition always holds for SInst-Gen inferences.

Proposition 1. *In any inference by SInst-Gen, the mgu θ is a proper instantiator for at least one of the literals L or L'.*

Proof. Literals L and $\overline{L}'$ are selected by sel. Therefore $L\bot$ and $\overline{L}'\bot$ are true in the model $I_\bot$ of $S\bot$. Assume that θ is not proper for both L and L'. Then, $L\bot = L\theta\bot = L'\theta\bot = L'\bot$ contradicting that both $L\bot$ and $\overline{L}'\bot$ are true in $I_\bot$. ❑

Selection functions can dramatically restrict the applicability of the inferences.

Example 2. Let S be the following set of clauses:

$$A(x_1, x_2, y) \vee A(x_2, x_3, y) \vee \ldots \vee A(x_n, x_{n+1}, y) \qquad (1)$$
$$\neg A(c_1, d, y) \vee \neg A(c_2, d, y) \vee \ldots \vee \neg A(c_n, d, y). \qquad (2)$$

Unrestricted applications of Inst-Gen will generate exponentially many (wrt. n) different instances of the first clause. Indeed, it is easy to see that using Inst-Gen one

can derive every instance of (1) where each variable x_i with an odd index i is mapped into one of the constants $c_1, \ldots, c_n$ and variables with even indexes are mapped into d. There are exponentially many such instances. Let us now consider SInst-Gen on this set of clauses. Consider a model $I_\bot$ of $S\bot$. Assume that $I_\bot \models A(\bot, \bot, \bot)$ and $I_\bot \models \neg A(c_1, d, \bot)$. Let sel be a selection function based on $I_\bot$ selecting $A(x_1, x_2, y)$ in clause (1) and $\neg A(c_1, d, y)$ in clause (2). Applying SInst-Gen to clauses (1) and (2) we obtain the conclusion:

$$A(c_1, d, y) \vee A(d, x_3, y) \vee \ldots \vee A(x_n, x_{n+1}, y). \qquad (3)$$

Now, extending the model $I_\bot$ to satisfy $A(d, \bot, \bot)$ and the selection function to select $A(d, x_3, y)$ in (3) will block all further inferences by SInst-Gen. The completeness Theorem 2, below, implies that S is satisfiable. This example is also interesting because most state-of-the-art resolution-based provers (e.g., E, Metis, SPASS and Vampire in the resolution mode) do not terminate on this set of clauses already for $n = 7$.

A natural generalisation of SInst-Gen is to consider hyper-inferences. For this we need to extend selection functions to select sets of literals from a clause rather than one literal. More formally, let S be a set of clauses such that $S\bot$ is consistent and $I_\bot$ a propositional model of $S\bot$. A *hyper-selection function* hsel for a set of clauses S is a mapping from clauses in S to multisets of literals such that $\emptyset \neq \mathsf{hsel}(C) \subseteq C$ for each clause $C \in S$. Literals L in $\mathsf{hsel}(C)$ are called *selected* in C (by hsel). For each clause $C \in S$ let us define a multiset of literals $\mathsf{sat}_\bot(C) = \{L \in C \mid I_\bot \models L\bot\}$. We say that a hyper-selection function hsel is *based* on $I_\bot$ if $\mathsf{hsel}(C) \subseteq \mathsf{sat}_\bot(C)$ for every $C \in S$. Thus, hyper-selection functions select some or all of the literals in a clause, whose $\bot$-instances are true in $I_\bot$. Let hsel be a hyper-selection function based on a model $I_\bot$ of $S\bot$. Instance generation, based on hsel, is defined as follows.

SHInst-Gen

$$\frac{\overline{L}'_1 \vee C_1 \quad \ldots \quad \overline{L}'_k \vee C_k \quad L_1 \vee \ldots \vee L_k \vee D}{(\overline{L}'_1 \vee C_1)\theta \quad \ldots \quad (\overline{L}'_k \vee C_k)\theta \quad (L_1 \vee \ldots \vee L_k \vee D)\theta}$$

where (i) θ is the most general unifier of $(L_1, L'_1), \ldots, (L_k, L'_k)$, and
(ii) $\mathsf{hsel}(L_1 \vee \ldots \vee L_k \vee D) = \{L_1, \ldots, L_k\}$, and
(iii) $\overline{L}'_i \in \mathsf{hsel}(\overline{L}'_i \vee C_i)$, for $1 \leq i \leq k$.

It is easy to see that SInst-Gen is a special case of SHInst-Gen when the hyper-selection function is restricted to select exactly one literal in each clause. As in the case of SInst-Gen, conditions on selection functions imply that θ is a proper instantiator for each pairs of literals (L_i, L'_i), $1 \leq i \leq k$.

Consider a set of clauses S such that $S\bot$ is satisfiable and a hyper-selection function hsel based on a model of $S\bot$. A set of clauses S is *SHInst-Gen saturated* wrt. hsel if the conclusion of any SHInst-Gen inference with premises in S is also in S. Now we can formulate the completeness theorem for SHInst-Gen, which also applies to SInst-Gen as a special case.

Theorem 2. [18] *Let S be a set of clauses such that $S\bot$ is satisfiable. If S is SHInst-Gen saturated wrt. a hyper-selection function based on a model of $S\bot$ then S is satisfiable.*

5 Redundancy Elimination

Redundancy elimination is crucial for practical applicability of any calculus. Our framework allows one to formulate a semantic-based notion of redundant clauses and redundant inferences. We first formulate redundancy notions for ground closures, which play a similar role to ground clauses in the resolution calculus.

Let $\succ$ be a closure ordering. Consider a set of ground closures U. A ground closure $C \cdot \sigma$ is called *redundant* in U if there exist ground closures $C_1 \cdot \sigma_1, \ldots, C_k \cdot \sigma_k$ in U such that, (1) $C_1 \cdot \sigma_1, \ldots, C_k \cdot \sigma_k \models C \cdot \sigma$ and (2) $C \cdot \sigma \succ C_i \cdot \sigma_i$ for each $0 \leq i \leq k$. In other words, a ground closure is redundant in U if it logically follows from smaller closures (w.r.t. $\succ$) in U.

We adapt this redundancy notion to be defined also on clauses by observing that a clause C is representing the set of all its ground closures $C \cdot \sigma$. Let S be a set of clauses and $\widehat{S}$ the set of all ground closures represented by clauses in S. A clause C (possibly non-ground) is called redundant in S if each ground closure $C \cdot \sigma$ is redundant in $\widehat{S}$. This abstract redundancy criterion can be used to justify many standard redundancies.

Tautologies. Note that tautologies are implied by the empty set of closures and therefore are redundant in any set of clauses.

Subsumption. A clause C *strictly subsumes* a clause C' if there is a substitution θ such that $C\theta \subsetneq C'$. For example, $A(x, y)$ strictly subsumes $A(x, a) \vee B(x)$. An ordering $\succ$ on ground clauses is called *strict subsumption compatible* if $C' \succ C$ for each ground clauses C and C' such that C is a strict sub-multiset of C', i.e., $C \subsetneq C'$. A closure ordering is *strict subsumption compatible* if its restriction to ground clauses is strict subsumption compatible. It is easy to see that if the closure ordering $\succ$ is strict subsumption compatible, then strict subsumption is an admissible redundancy. One can also eliminate non-strictly subsumed clauses in the case of non-proper instantiators as follows. An ordering is called *non-proper subsumption compatible* if $C' \cdot \sigma \succ C \cdot \theta\sigma$ for every closures $C' \cdot \sigma$ and $C \cdot \theta\sigma$ where $C\theta = C'$ and θ is a non-proper instantiator and not a renaming. For example, if the closure ordering $\succ$ is non-proper subsumption compatible, then $A(x, x) \vee B(x)$ is redundant w.r.t. $A(x, y) \vee B(y)$. Closure orderings which are strict and at the same time non-proper subsumption compatible are called *subsumption compatible orderings*. It is easy to see that any subsumption compatible ordering on ground clauses can be extended to a subsumption compatible closure ordering. Let us note that full subsumption is not an admissible redundancy. Indeed, all clauses derived by Inst-Gen are subsumed by the initial clauses.

Instantiation-specific redundancy. Our semantic redundancy criteria can also be used to define instantiation-specific redundancies. In particular, consider a clause C and a proper instance of C, $D = C\theta$. Then, all ground closures of C, which are also represented by D, are redundant. We will discuss this redundancy in detail when we consider dismatching constraints in Section 6.

An inference with premises $C_1, \ldots, C_n$ and a unifier θ (thus deriving conclusions $C_1\theta, \ldots, C_n\theta$) is *redundant* in S if for every substitution ρ grounding all the $C_i\theta$ there exists an index i_0 such that $C_{i_0} \cdot \theta\rho$ is redundant in S.

Example 3. Let the set of function symbols in Σ consists of the constants a and b and assume that $\succ$ is subsumption compatible. Consider the following set of clauses:

$$\underline{A(x,y)} \vee B(y) \quad (1) \qquad \underline{B(a)} \quad (2)$$
$$\neg\underline{A(a,z)} \vee C(z) \quad (3) \qquad \underline{C(b)} \quad (4)$$

Let us show that the SInst-Gen inference between clauses (1) and (3) is redundant. Let $\theta = \{a/x; z/y\}$ be the mgu of atoms of the selected literals in clauses (1) and (3). Then, for any grounding substitution ρ, either closure $(A(x,y) \vee B(y)) \cdot \theta\rho$ is redundant (in the case $z\rho = a$), or closure $(\neg A(a,z) \vee C(z)) \cdot \theta\rho$ is redundant (in the case $z\rho = b$).

An important property of the (SH)Inst-Gen calculus is that adding the conclusion of an inference makes the inference redundant. The next proposition shows that in order to make an inference redundant it is sufficient to add to the clause set at least one properly instantiated clause from the conclusion (such a clause always exits, by Proposition 1).

Proposition 2. *Let $C\theta$ be a conclusion of an (SH)Inst-Gen inference and a proper instance of its respective premise C. If $C\theta$ is in S, or is redundant in S, then the inference is redundant.*

Proof. Immediately follows from the definition of redundant inference.

We can have additional flexibility with partial instantiations as follows. Let $C\theta$ be as in Proposition 2 and θ' any substitution which is a proper instantiator for C and more general than θ. Then, adding $C\theta'$ to S makes the inference redundant.

Example 4. Consider the following set of clauses $S = \{C, D\}$ where $C = \underline{A(x,y)} \vee B(x)$ and $D = \neg\underline{A(f(a),b)}$. The most general unifier of $A(x,y)$ and $A(\overline{f(a),b})$ is $\theta = \{f(a)/x, b/y\}$. Consider a proper instantiator of C which is more general than θ, for example $\theta' = \{f(z)/x\}$. Then, adding $C\theta' = A(f(z),y) \vee B(f(z))$ to S makes the inference between C and D redundant. Such partial instantiations can be useful for keeping reasoning at a more general level and can be combined with dismatching constraints (see Section 6).

The idea of partial instantiations is developed further in [18], where it is used to approximate first-order clauses by clauses from first-order fragments beyond propositional logic.

A set of clauses S is called *SHInst-Gen saturated up to redundancy* if all inferences in SHInst-Gen from premises in S are redundant in S. The next theorem shows that completeness is preserved under redundancy elimination.

Theorem 3. [18] *Let S be a set of clauses such that $S\bot$ is satisfiable. Let* hsel *be a hyper-selection function based on a model of $S\bot$. If S is SHInst-Gen saturated up to redundancy then S is satisfiable.*

This theorem also applies to weaker systems SInst-Gen and Inst-Gen. We show below that Theorem 3 is a consequence of a more general Theorem 4.

For a finer control over redundancy we consider the notion of inferences and saturation at the level of ground closures. For a set of ground closures U let $\overline{U}$ denote the

set of clauses C such that $C \cdot \sigma$ is in U for some grounding substitution σ. Let hsel be a hyper-selection function for $\overline{U}$ based on a model $I_\bot$ of $\overline{U}\bot$. A SHInst-Gen inference on ground closures is defined as follows.

SHInst-Gen (ground closures)

$$\frac{(\overline{L}_1' \vee C_1)\cdot\sigma_1 \quad \ldots \quad (\overline{L}_k' \vee C_k)\cdot\sigma_k \quad (L_1 \vee \ldots \vee L_k \vee D)\cdot\sigma}{(\overline{L}_1' \vee C_1)\theta\cdot\tau_1 \quad \ldots \quad (\overline{L}_k' \vee C_k)\theta\cdot\tau_k \quad (L_1 \vee \ldots \vee L_k \vee D)\theta\cdot\tau}$$

where (i) θ is the most general unifier of $(L_1, L_1'),\ \ldots,\ (L_k, L_k')$,
(ii) $\mathsf{hsel}(L_1 \vee \ldots \vee L_k \vee D) = \{L_1, \ldots, L_k\}$, and
(iii) $\overline{L}_i' \in \mathsf{hsel}(\overline{L}_i' \vee C_i)$, for $1 \leq i \leq k$, and
(iv) $(\overline{L}_1' \vee C_1)\sigma_1 = (\overline{L}_1' \vee C_1)\theta\tau_1, \ldots, (\overline{L}_k' \vee C_k)\sigma_k = (\overline{L}_k' \vee C_k)\theta\tau_k$,
$(L_1 \vee \ldots \vee L_k \vee D)\sigma = (L_1 \vee \ldots \vee L_k \vee D)\theta\tau$

A ground SHInst-inference with premises $C_1\cdot\sigma_1, \ldots, C_n\cdot\sigma_n$ and conclusion $C_1\theta\cdot\tau_1, \ldots, C_n\theta\cdot\tau_n$ is redundant in a set of ground closures U if at least one of the closures $C_i\cdot\theta\tau_i = C_i\cdot\sigma_i$ is redundant in U, for $1 \leq i \leq n$. We say that a set of ground closures U is *SHInst-Gen saturated up to redundancy* wrt. hsel , if any ground SHInst-inference with a premise in U is redundant in U.

Theorem 4. *Let U be a set of ground closures such that $\overline{U}\bot$ is satisfiable. Let* hsel *be a hyper-selection function based on a model of $\overline{U}\bot$. If U is SHInst-Gen saturated up to redundancy wrt.* hsel *then U is satisfiable.*

Proof. For simplicity of exposition we first prove the theorem for a special case of binary inferences (SInst-Gen) and later show how to modify the proof for hyper-inferences (SHInst-Gen). The proof is based on an adaptation of the model-generation technique (see [3, 38]).

Let U be a set of ground closures such that $\overline{U}\bot$ is satisfiable. First, we construct a candidate (partial) model I_U of U and then show that if U is SInst-saturated up to redundancy, any total extension of I_U is indeed a model of U. Let $I_\bot$ be a model of $\overline{U}\bot$ and sel a selection function on clauses in $\overline{U}$ based on $I_\bot$.

Informally, we construct the model I_U by adding literals in a way to satisfy closures in U. In order to construct I_U we construct a sequence of partial models and sets of literals by induction on closures ordered by $\succ$ as follows. Let $\widehat{C} = C\cdot\sigma$ be a ground closure. Suppose, as an induction hypothesis, we have defined sets of literals $\epsilon_{\widehat{D}}$, for all ground closures $\widehat{D}$ smaller than $\widehat{C}$ wrt. $\succ$. Let $I_{\widehat{C}}$ denote the set $\bigcup_{\widehat{C}\succ\widehat{D}} \epsilon_{\widehat{D}}$.

We define $\epsilon_{\widehat{C}}$ as follows. Assume $\widehat{C}$ is in U and define $L = \mathsf{sel}(C)$. We define $\epsilon_{\widehat{C}} = \{L\sigma\}$, if the following conditions hold:

1. $I_{\widehat{C}} \not\models C\sigma$, i.e., there is a total model extending $I_{\widehat{C}}$ in which $C\sigma$ is false, and
2. $L\sigma$ is undefined in $I_{\widehat{C}}$, i.e., neither $I_{\widehat{C}} \models L\sigma$ nor $I_{\widehat{C}} \models \overline{L}\sigma$ holds.

Otherwise, if either $\widehat{C}$ is not in U or at least one of the conditions (1)–(2) is not satisfied, we define $\epsilon_{\widehat{C}} = \emptyset$. In the case when $\epsilon_{\widehat{C}} = \{L\sigma\}$ we say that $L\sigma$ is *produced* by $\widehat{C}$. Define $I_U = \bigcup_{\widehat{C}} \epsilon_{\widehat{C}}$. It follows immediately from the construction that I_U is consistent.

Now, let us assume that U is SInst-saturated up to redundancy. Let I be any total extension of I_U. In order to prove our theorem we show that I is a model of U.

First, we note that our model construction satisfies the following:

- *monotonicity:* if a ground closure $\widehat{C}$ is true in some $I_{\widehat{C}}$ then $\widehat{C}$ true in all $I_{\widehat{C}'}$ for $\widehat{C}' \succ \widehat{C}$ and also true in I, and
- *productiveness:* if $\widehat{C}$ is a productive closure then (i) $I_{\widehat{C}} \not\models \widehat{C}$, and (ii) $\widehat{C}$ is true in $I_{\widehat{C}} \cup \epsilon_{\widehat{C}}$ and hence $\widehat{C}$ true in I.

Now, by induction on $\succ$ we show that every ground closure $\widehat{C}$ in U is true in $I_{\widehat{C}} \cup \epsilon_{\widehat{C}}$. From this and monotonicity of the model construction our theorem follows. Assume otherwise. Let $\widehat{C} = C \cdot \sigma$ be the minimal ground closure in U such that $I_{\widehat{C}} \cup \epsilon_{\widehat{C}} \not\models \widehat{C}$. Let $L = \mathsf{sel}(C)$. As $\widehat{C}$ is not productive and $I_{\widehat{C}} \not\models \widehat{C}$ we have $\overline{L}\sigma \in I_{\widehat{C}}$. Indeed, otherwise all conditions (1)–(2) of the model construction would be satisfied and $\widehat{C}$ would be productive. Let $\widehat{D} = D \cdot \tau$ be a closure producing $\overline{L}\sigma$ into $I_{\widehat{C}}$, where $\widehat{C} \succ \widehat{D}$. We have $D = \overline{L}' \vee D'$ where $\overline{L}'$ is selected by sel and $\overline{L}'\tau = \overline{L}\sigma$. Therefore, a ground SInst-Gen inference is applicable to closures $\widehat{C}$ and $\widehat{D}$ producing $C\theta \cdot \sigma'$ and $D\theta \cdot \tau'$, where (i) θ is the most general unifier of L and L', and (ii) $C\theta\sigma' = C\sigma$ and $D\theta\tau' = D\tau$. By the assumption of the theorem U is SInst-saturated and hence this inference is redundant. Therefore, at least one of the closures $C \cdot \theta\sigma' = C \cdot \sigma = \widehat{C}$ or $D\theta \cdot \tau' = D \cdot \tau = \widehat{D}$ is redundant. Assume that $\widehat{C}$ is redundant. Then, $\widehat{C}$ follows from smaller (wrt. $\succ$) closures $\widehat{C}_1, \ldots, \widehat{C}_n$ in U. By induction hypothesis, we have that each $\widehat{C}_i$ is true in $I_{\widehat{C}_i} \cup \epsilon_{\widehat{C}_i}$, and by monotonicity is true in $I_{\widehat{C}}$, for $1 \leq i \leq n$. From this it follows that $\widehat{C}$ is true in $I_{\widehat{C}}$, contradicting our assumption. Similarly, we arrive at a contradiction when we assume that $\widehat{D}$ is redundant. Indeed, if $\widehat{D}$ follows from smaller closures in U, then by induction hypothesis these closures are true in $I_{\widehat{D}}$, hence $\widehat{D}$ is true in $I_{\widehat{D}}$, contradicting productiveness of $\widehat{D}$. This concludes the proof of this theorem for the case of SInst-Gen.

The case of hyper-inferences is similar, we only need to make the following modifications: (i) consider hyper-selection hsel in place of selection sel, (ii) in the model construction, we define $\epsilon_{\widehat{C}} = \{L\sigma\}$ if $\widehat{C}$ is in U and $L \in \mathsf{hsel}(C)$ such that conditions (1)–(2) are satisfied; if there are several such literals we can choose any of them to define $\epsilon_{\widehat{C}}$, (iii) we prove that I is a model for U in a similar way as above. ❑

Let us note that completeness Theorem 3 for clauses follows from Theorem 4 as follows. Let S be a set of clauses such that $S\bot$ is satisfiable. Assume that S is SHInst-Gen saturated up to redundancy wrt. a hyper-selection function hsel based on a model of $S\bot$. Let U be the set of all ground instances of clauses in S. We have that $\overline{U} = S$ and U is SHInst-Gen saturated up to redundancy wrt. hsel. By Theorem 4, U is satisfiable and therefore S is satisfiable.

Next we describe how our abstract redundancy criteria can be used to justify practical redundancy elimination. We start by introducing dismatching constraints which are used to discard redundant ground closures. Then we show how the reasoner for ground clauses can be used to simplify clauses. Finally we show how the resolution calculus can be combined with instantiation.

6 Dismatching Constraints

Let us consider a clause $C \in S$. As we have seen, adding a proper instance $C\theta$ of a clause C (e.g., as a result of applying an SInst-Gen inference) makes some ground closures represented by C redundant and consequently certain inferences with C redundant. In particular, all closures $C \cdot \sigma$ such that $C\sigma = C\theta\tau$ for a grounding substitution τ are redundant in the presence of $C\theta$. We can efficiently represent this information about redundant closures using dismatching constraints. Let us note that in the context of resolution and paramodulation various kinds of constraints have been considered (see e.g. [10, 29, 38]). Dismatching constraints are particularly attractive: on the one hand they provide powerful restrictions for the instantiation calculus, and on the other hand, checking dismatching constraints can be efficiently implemented.

An *atomic dismatching constraint* is a pair of variable disjoint tuples of terms, denoted $\langle s_1, \ldots, s_n \rangle \not\trianglelefteq \langle t_1, \ldots, t_n \rangle$, or simply $\bar{s} \not\trianglelefteq \bar{t}$. A solution to a constraint $\bar{s} \not\trianglelefteq \bar{t}$ is a substitution σ such that for every substitution γ, $\bar{s}\sigma \not\equiv \bar{t}\gamma$. For example, consider an atomic dismatching constraint $\varphi(x, y) = \langle x \rangle \not\trianglelefteq \langle f(y) \rangle$. Then, the substitution $\sigma_1 = \{a/x\}$ is a solution to $\varphi(x, y)$, but $\sigma_2 = \{f(g(a))/x\}$ is not since there is a substitution $\gamma = \{g(a)/y\}$ such that $\langle x \rangle \sigma_2 \equiv \langle f(y) \rangle \gamma$. It is easy to see that an atomic dismatching constraint $\bar{s} \not\trianglelefteq \bar{t}$ is satisfiable if and only if for all substitutions γ, $\bar{s} \not\equiv \bar{t}\gamma$. In other words, an atomic dismatching constraint $\bar{s} \not\trianglelefteq \bar{t}$ is not satisfiable if and only if there is a substitution γ such that $\bar{s} \equiv \bar{t}\gamma$, which is a familiar matching problem.

A *dismatching constraint* $ds(\bar{s}, \bar{t}) = \wedge_{i=1}^{n} \bar{s}_i \not\trianglelefteq \bar{t}_i$ is a conjunction of atomic dismatching constraints where every $\bar{t}_i$ is variable disjoint from all $\bar{s}_j$, and $\bar{t}_k$, for $i \neq k$. A substitution σ is a *solution* of a dismatching constraint $\wedge_{i=1}^{n} \bar{s}_i \not\trianglelefteq \bar{t}_i$ if σ is a solution of each $\bar{s}_i \not\trianglelefteq \bar{t}_i$, for $1 \leq i \leq n$.

Proposition 3. *The satisfiability problem for dismatching constraints can be solved in linear-time.*

Proof. As we noted above, the satisfiability problem for dismatching constraints can be reduced in linear-time to the matching problem which can be solved in linear-time (see, e.g., [2]). ❑

A *constrained clause* $C \mid [\ \varphi\ ]$ is a clause C together with a dismatching constraint φ. We will always assume that for a constrained clause $C \mid [\ \wedge_{i=1}^{n} \bar{s}_i \not\trianglelefteq \bar{t}_i\ ]$, the clause C is variable disjoint from all t_i, $1 \leq i \leq n$. A constrained clause $C \mid [\ \varphi\ ]$ represents the set of ground closures $\{C \cdot \sigma \mid \sigma \text{ is a solution to } \varphi\}$, denoted $Cl(C \mid [\ \varphi\ ])$. An unconstrained clause C can be seen as a constrained clause with the empty constraint $C \mid [\ ]$. For a set S of constrained clauses, $Cl(S)$ denotes the set of all ground closures represented by constrained clauses in S. Let S be a set of constrained clauses, then $\widetilde{S}$ denotes the set of all *unconstrained clauses* obtained from S by dropping all constraints. We say that a set of constrained clauses S is *well-constrained* if $Cl(S) \models Cl(\widetilde{S})$. In the following we consider only well-constrained sets of clauses.

Now we formulate an extension of SInst-Gen with dismatching constraints, called DSInst-Gen. For simplicity of the exposition we consider only binary inferences, the extension to hyper-inferences can be done in a similar way. DSInst-Gen inferences will generate new instances of clauses and also extend constraints of clauses in the premises.

Let S be a set of constrained clauses such that $\widetilde{S}\bot$ is consistent and sel be a selection function based on a model $I_\bot$ of $\widetilde{S}\bot$. Then, DSInst-Gen inference system is defined as follows.

DSInst-Gen

$$\frac{L \vee C \mid [\, \varphi \,] \quad \overline{L}' \vee D \mid [\, \psi \,]}{L \vee C \mid [\, \varphi \wedge \bar{x} \not\trianglelefteq \bar{x}\theta \,] \quad (L \vee C)\theta}$$

where (i) $\bar{x}$ is a tuple of all variables in L, and
(ii) θ is the most general unifier of L and L', wlog. we assume that variables in the range of θ do not occur in $L \vee C \mid [\, \varphi \,]$ and the domain of θ contains all variable in $\bar{x}$, and
(iii) $\mathsf{sel}(L \vee C) = L$, and $\mathsf{sel}(\overline{L}' \vee D) = \overline{L}'$, and
(iv) θ is a proper instantiator for L, and
(v) $\varphi\theta$ and $\psi\theta$ are both satisfiable dismatching constraints.

DSInst-Gen is a replacement rule, that is replacing the clause in the left premise by clauses in the conclusion. The clause in the right premise can be seen as a side condition, that is no instances of this clause are produced.

We can see that in addition to semantic restrictions imposed by the selection function, instantiation rule is applicable only if dismatching constraints are satisfiable after applying θ. Let us note that applications of DSInst-Gen preserves well-constrainedness of sets of clauses.

The notion of redundancy can be easily adapted from clauses to constrained clauses as dismatching constraints can be seen as a method for discarding redundant ground closures. A constrained clause $C \mid [\, \varphi \,]$ is *redundant* wrt. a set of constrained clauses S if all closures in $Cl(C \mid [\, \varphi \,])$ are redundant in $Cl(S)$. A DSInst-Gen inference with the premises $C \mid [\, \varphi \,]$, $D \mid [\, \psi \,]$ and the conclusion $C \mid [\, \varphi \wedge \bar{x} \not\trianglelefteq \bar{x}\theta \,]$, $C\theta$ is *redundant* in S if the following holds. For any substitution ρ grounding for $C\theta$ and $D\theta$, which is a solution to $\varphi\theta$ and $\psi\theta$, either $C \cdot \theta\rho$ or $D \cdot \theta\rho$ is redundant in $Cl(S)$. A set of constrained clauses S is DSInst-Gen *saturated up to redundancy* if all inferences by DSInst-Gen from premises in S are redundant in S. The DSInst-Gen calculus can be seen as a way of lifting the (binary version of) SHInst-Gen calculus from closures to constrained clauses.

Theorem 5. *Let S be a set of constrained clauses such that $\widetilde{S}\bot$ is satisfiable. If S is DSInst-Gen saturated up to redundancy wrt. a selection function based on a model of $\widetilde{S}\bot$, then $Cl(S)$ is satisfiable.*

Proof. Indeed, if S satisfies the assumption of the theorem then $Cl(S)$ is SHInst-Gen saturated up to redundancy. Therefore the theorem follows from Theorem 4.

DSInst-Gen saturation strategies will be considered in Section 9.

Example 5. Let S be the following set of clauses where selected literals are underlined.

$$\underline{\neg A(x)} \vee C(x) \quad (0), \qquad \begin{array}{rcll} \underline{A(f(y))} & \vee & D_1 & (1), \\ \underline{A(f^{i_2}(y))} & \vee & D_2 & (2), \\ & \cdots & & \\ \underline{A(f^{i_n}(y))} & \vee & D_n & (n). \end{array}$$

Where $i_k \geq 1$ for $2 \leq k \leq n$, and $f^m(t)$ denotes m applications of f: $f(\ldots f(t)\ldots)$. Applying DSInst-Gen to clauses (0) and (1) will produce $\neg A(f(x)) \vee C(f(x))$, denoted as $(0'')$. We also replace clause (0) with $\neg A(x) \vee C(x) \mid [\, x \not\trianglelefteq f(z)\,]$, denoted as $(0')$, obtaining a new set of clauses S'. Assume that the new selection for S' is the same on the old clauses $(1), \ldots, (n)$ and $(0')$ inherits the selection from (0). This implies that $C(x)$ should be selected in $(0'')$. Therefore S' will be as follows:

$$\begin{array}{rcl} \underline{\neg A(x)} \vee C(x) \mid [\, x \not\trianglelefteq f(z)\,] & (0'), & \underline{A(f(y))} \vee D_1 \quad (1), \\ \neg A(f(x)) \vee \underline{C(f(x))} & (0''), & \underline{A(f^{i_2}(y))} \vee D_2 \quad (2), \\ & & \cdots \\ & & \underline{A(f^{i_n}(y))} \vee D_n \quad (n). \end{array}$$

We can see that S' is DSInst-Gen saturated and therefore S is satisfiable by Theorem 5. Indeed, inferences between clauses $(0')$ and (1)–(n) are blocked by the dismatching constraint of the clause $(0')$. Let us note that without dismatching constraints, we would need to consider all inferences between clauses (0) and (1)–(n).

Let us compare DSInst-Gen to resolution, assuming that the same selected literals are eligible for resolution inferences. First we note that all inferences between (0) and (1)–(n) are applicable. Keeping in mind that a clause can be seen as a representation of all its ground instances we can note that instances represented by $C(x)$ are copied at each resolution inference and recombined with different clauses. This can result in repeated work on the same instances of $C(x)$ and is known as the recombination problem [35]. Dismatching constraints allow one to avoid such problems in the instantiation setting.

Without losing completeness of DSInst-Gen we can replace satisfiability for dismatching constraints with a stronger notion, called ground satisfiability. We say that a dismatching constraint φ is *ground satisfiable* if there is a grounding substitution σ which is a solution to φ. Obviously, if a constraint is ground satisfiable then it is satisfiable but converse need not hold. Consider a signature consisting of a constant a, a unary function symbol f and predicate symbols. Then, a constraint $x \not\trianglelefteq f(y) \wedge x \not\trianglelefteq a$ is satisfiable but not ground satisfiable. We can see that ground satisfiability is signature dependent, and two notions of satisfiability coincide in the case of signatures containing infinite number of constants. Problems related to ground satisfiability of dismatching constraints were studied in a number of works [27, 34, 42]. In contrast to the problem of satisfiability, which can be solved in linear-time, the problem of ground satisfiability is NP-complete.

Theorem 6. [27, 34] *The ground satisfiability problem for dismatching constraints is NP-complete.*

7 Simplification by Propositional Reasoning

Having at hand a powerful propositional solver it is natural to investigate methods for redundancy elimination which utilise propositional or ground reasoning.

Let us first consider the case of simplifying ground closures. In order to apply our abstract redundancy criterion to simplify a ground closure $C \cdot \sigma$ wrt. a set of ground closures U we consider a set of closures $Sim = \{D_1 \cdot \tau_1, \ldots D_n \cdot \tau_n\}$ (not necessarily contained in U) such that:

1. $U \models D_1 \cdot \tau_1, \ldots, U \models D_n \cdot \tau_n$, and
2. $D_1 \cdot \tau_1, \ldots, D_n \cdot \tau_n \models C \cdot \sigma$, and
3. $C \cdot \sigma \succ D_1 \cdot \tau_1, \ldots, C \cdot \sigma \succ D_n \cdot \tau_n$.

We call Sim a *simplification set* for $C \cdot \sigma$ wrt. U. If Sim is a simplification set for $C \cdot \sigma$ wrt. U then we can replace $C \cdot \sigma$ in U by closures in Sim, without losing neither soundness nor completeness. There are a number of issues to consider in this general scheme:

- how to choose a candidate for a simplification set Sim,
- how to check whether conditions (1)–(3) above are satisfied.

In this paper we consider the case when a candidate for a simplification set for a closure $C \cdot \sigma$ consists of a strict subclosure of $C \cdot \sigma$. A closure $D \cdot \tau$ is a *strict subclosure* of a closure $C \cdot \sigma$, denoted by $D \cdot \tau \subsetneq C \cdot \sigma$, if $D \subsetneq C$ and $D\tau \subsetneq C\sigma$. In this case, condition (2) is trivially satisfied. In order to satisfy condition (3) we assume that $\succ$ is subsumption compatible (see Section 5). The most difficult is to check condition (1). Indeed, condition (1) is at least as complex as the initial problem of unsatisfiability of U. Fortunately, for redundancy elimination it is sufficient to consider sound approximations of the entailment relation in (1). Next we consider such approximations based on propositional reasoning.

Let us put these considerations in the context of constrained clauses. Let S be a set of (well-constrained) clauses. The notion of a simplification set can be readily adapted for clauses. Together with S, we consider a set of ground clauses S_{gr} such that $S \models S_{gr}$. For simplicity of exposition we assume that S_{gr} is an extension of $\widetilde{S}\bot$ by auxiliary ground clauses implied by S. The set S_{gr} will be used in propositional reasoning for approximating condition (1). Let us note that clauses in S_{gr} do not participate in instantiation inferences. In Section 7.1 we consider simplification of ground clauses, and in Section 7.2 simplification of clauses with variables.

7.1 Simplification of ground clauses

In this section we consider the case of simplifying ground clauses wrt. a set of clauses S. Let C be a ground clause to simplify. As a candidate for a simplification set we consider a set consisting of a strict subclause $D \subsetneq C$. Using the propositional solver we can check whether $S_{gr} \models D$. If this is the case, adding D to S makes C redundant wrt. to the new set $S \cup \{D\}$. We call this simplification as *global propositional subsumption* wrt. S_{gr}.

Global propositional subsumption

$$\frac{D \vee D'}{D}$$

where $S_{gr} \models D$ and D' is not empty.

Global propositional subsumption is a simplification rule, which allows one to remove the clause in the premise after adding the conclusion. Let us note that although the number of possible subclauses is exponential wrt. the number of literals, in a linear number of implication checks we can find a minimal wrt. inclusion subclause $D \subsetneq C$ such that $S_{gr} \models D$ or show that such a subclause does not exist.

Let us show that global propositional subsumption generalises a number of usual redundancy eliminations. First, note that global propositional subsumption generalises strict propositional subsumption. Indeed, if there is a strict subclause $D \subsetneq C$ such that $D \in S$ then $S_{gr} \models D$ and therefore C is globally subsumed by S_{gr}. Next, we consider *propositional subsumption resolution*.

Propositional subsumption resolution

$$\frac{L \vee D' \qquad \overline{L} \vee D \vee D'}{D \vee D'}$$

Propositional subsumption resolution is a simplification rule, which allows one to remove the right premise after adding the conclusion. Let us show that global subsumption generalises subsumption resolution. Indeed, if the premise clauses $L \vee D'$ and $\overline{L} \vee D \vee D'$ of subsumption resolution are in S then $S_{gr} \models D \vee D'$ and therefore $\overline{L} \vee D \vee D'$ is globally subsumed by S_{gr}.

In general, global subsumption involves reasoning with the whole set S_{gr}. For example, let S_{gr} contain the following clauses

$$A(f(\bot)) \vee B(g(c));\ \neg B(g(c)) \vee A(c);\ \neg A(f(\bot)) \vee A(c)$$

Then, a clause $A(c) \vee B(f(c))$ can be simplified to $A(c)$ wrt. to S_{gr}. In the cases we consider here, the clauses we try to simplify always follow from the set S (e.g. obtained by sound derivations from the initial set of clauses). Therefore, the clause we simplify, itself can be added to S_{gr} before simplification. For example, if we want to simplify a clause $C = \neg A(c) \vee B(f(c))$ wrt. S_{gr} above, we can first add C to S_{gr} obtaining $S'_{gr} = S_{gr} \cup \{C\}$ and then C can be simplified to $B(f(c))$ wrt. S'_{gr}.

7.2 Simplification of non-ground clauses

In this section we consider the case of simplifying non-ground clauses wrt. a set of clauses S, utilising propositional reasoning. For this we need soundly approximate semantic entailment. The approximation of entailment we use will be based on the following proposition.

Proposition 4. *Let $\varphi(\bar{x})$ and $\psi(\bar{x})$ be first-order formulas over a signature Σ and $\bar{c}$ a tuple of pairwise different constants not in Σ. Then, $\varphi(\bar{c}) \models \psi(\bar{c})$ implies $\forall \bar{x} \varphi(\bar{x}) \models \forall \bar{x} \psi(\bar{x})$.*

Proof. We have the following sequence of equivalences and implications:

$$\begin{array}{l}\varphi(\bar{c}) \models \psi(\bar{c}) \Leftrightarrow \\ \models \varphi(\bar{c}) \rightarrow \psi(\bar{c}) \Leftrightarrow \\ \models \forall \bar{x}(\varphi(\bar{x}) \rightarrow \psi(\bar{x})) \Rightarrow \\ \models (\forall \bar{x}\varphi(\bar{x})) \rightarrow (\forall \bar{x}\psi(\bar{x})) \Leftrightarrow \\ \forall \bar{x}\varphi(\bar{x}) \models \forall \bar{x}\psi(\bar{x}).\end{array}$$

❑

We can use Proposition 4 as follows. Let Σ_C be a signature consisting of an infinite number of constants not occurring in Σ. Let Ω be a set of injective substitutions mapping variables to constants in Σ_C. We call C' an Ω-instance of a clause C if $C' = C\gamma$ where $\gamma \in \Omega$. With each clause $C \in S$ we associate a set of Ω instances of C, denoted C_Ω. Let us assume that for every clause $C \in S$, $C_\Omega \subseteq S_{gr}$. Then, if we show that some Ω-instance of a given clause D is implied by S_{gr}, from Proposition 4 it follows that S implies D. Now we can formulate extension of global subsumption to the non-ground case:

Global subsumption (non-ground)

$$\frac{(D \vee D')\theta}{D}$$

where (i) θ is a (possibly identity) substitution, and
(ii) $S_{gr} \models D\gamma$ for some $\gamma \in \Omega$, and
(iii) D' is not empty.

As in the ground case, global subsumption is a simplification rule. Informally, in order to simplify a clause C using global subsumption it is sufficient to find a clause D strictly subsuming C, (i.e., $D\theta \subsetneq C$ for a substitution θ), such that an Ω-instance $D\gamma$ of D follows from S_{gr}. Then, adding D into S makes C redundant in S. There are several non-trivial issues to consider when we try to apply global subsumption. These are:

1. which Ω-instances of clauses in S to add to S_{gr},
2. which clause D to use as a candidate for the conclusion, and
3. which Ω-instances of the candidate clause D to check for entailment.

Since there are infinitely many possible Ω-instances of a clause, we restrict ourselves to some heuristics. Let us describe one of them. Assume that constants in Σ_C are ordered: $c_1, \ldots, c_k, \ldots$. For a given clause C, fix an ordering on variables occurring in C: $x_1, \ldots, x_n$. We define a substitution $\gamma_C : \{x_i \mapsto c_i \mid 1 \leq i \leq n\}$. Trivially $C\gamma_C$ is an Ω-instance of C. To address issue (1) above we assume that for a clause $C \in S$, $C\gamma_C$ is in S_{gr}. For (2), we choose candidates for the conclusion of global subsumption among strict subclauses of a given clause. For (3), we use $D\gamma_D$ as an Ω-instance of the candidate clause D.

Example 6. Consider the following example where S consists of the first four clauses of SYN-832 problem from the TPTP library [50]. For readability we rename predicate symbols.

$$A \quad (1) \qquad \neg A \vee B(x_1) \quad (2)$$
$$\neg B(x_1) \vee \neg A \vee C(x_1, x_2) \quad (3) \qquad \neg C(x_1, x_2) \vee \neg B(x_1) \vee \neg A \vee D(x_1, x_2, x_3) \quad (4)$$

These clauses come from translations of modal formulae [26]. The set S_{gr}, in addition to clauses from $S\bot$ will contain Ω-instances $G\gamma_G$ for $G \in S$:

$$A \quad (1) \qquad \neg A \vee B(c_1) \quad (2)$$
$$\neg B(c_1) \vee \neg A \vee C(c_1, c_2) \quad (3) \qquad \neg C(c_1, c_2) \vee \neg B(c_1) \vee \neg A \vee D(c_1, c_2, c_3) \quad (4)$$

Now, using global subsumption we can simplify clauses in S to the following set of unit clauses S':

$$A \quad (1) \qquad B(x_1) \quad (2)$$
$$C(x_1, x_2) \quad (3) \qquad D(x_1, x_2, x_3) \quad (4)$$

Let us emphasise that pure propositional reasoning suffices for these simplifications. In practice, we can employ efficient propositional solvers for such simplifications in a black-box fashion. One can exploit incrementality of state-of-the-art propositional solvers such as MiniSAT [15] which allow one to check satisfiability of sets of propositional clauses under assumed sets of literals. In order to check whether an Ω-instance $C\gamma = L_1\gamma \vee \ldots \vee L_n\gamma$ follows from a set of ground clauses S_{gr} it is sufficient to check unsatisfiability of S_{gr} under the assumption consisting of literals $\overline{L}_1\gamma, \ldots, \overline{L}_n\gamma$. Using linear search, one can find a minimal wrt. inclusion sub-clause of $C\gamma$ which follows from the set of ground clauses in less than n implications checks. Another approach for obtaining minimal implied sub-clauses can be based on minimal unsatisfiability cores returned by propositional solvers. Since in practice it is sufficient to approximate simplifications one can use efficient incomplete tests for checking propositional implications based, e.g., unit propagation or restricting the number of backjumps.

Let us note that global subsumption can be used not only in instantiation-based calculi, but in any calculi where strict subsumption is an admissible redundancy elimination such as, e.g., resolution. Simplifications by ground reasoning have been independently investigated in different settings (see, e.g., [1, 22, 30, 31]).

Generating implied clauses by propositional reasoning. Using propositional reasoning we can generate clauses implied by S_{gr}. Let $C\gamma$ be an Ω-instance of a clause C, such that $C\gamma$ is implied by S_{gr}. Then, from Proposition 4 it follows that C is implied by S. Therefore we can add C to S and use C for simplifications of clauses in S. In particular, we can use implied clauses for simplifications such as strict subsumption.

Let us note the main difference with global subsumption: in global subsumption we check implications of given clauses (e.g., strict subclauses of a clause to be simplified). Here we generate implied clauses on the fly, e.g., when we check consistency of S_{gr} and use obtained clauses later for simplifications. Most state-of-the-art propositional reasoners generate such clauses, called learnt clauses or lemmas, during the proof search. If a

propositional lemma $C\gamma$ is generated then $C\gamma$ is implied by S_{gr} and the corresponding first-order clause C is implied by S. We can use the obtained first-order lemma C for further simplifications.

To conclude, we have shown that propositional reasoning can be used not only to guide the instantiation process but also for simplification of clauses.

8 Combination of Instantiation with Resolution

One of the attractive properties of the instantiation calculus is that the number of literals in clauses does not increase during instantiation. On the other hand, if we consider the instantiation calculus without simplifications, the number of literals in the generated clauses does not decrease. Consequently, instantiation is not well-suited for generating clauses which can be used in simplifications, such as strict subsumption. We can overcome this limitation by combining instantiation with the (ordered) resolution calculus. There are different ways to combine instantiation with resolution, (see, e.g., [20, 36]), here we consider a simple one. We run resolution simultaneously with instantiation to generate additional clauses that can be used for simplifications. Let us note that clauses generated by resolution are used only for simplifications and do not participate in instantiation inferences. In addition, we can add Ω-instances of clauses generated by resolution to S_{gr} which in turn can be used for propositional-based simplifications discussed above.

Example 7. Consider the following set of clauses S:

$$\begin{array}{rll} \neg A(x) & \vee H(x) & (1) \\ A(f(x)) & \vee B(x) & (2) \\ \neg H(f(x)) & \vee B(x) & (3) \end{array}$$

Assume that in each clause (1)–(3) the first literal is eligible for resolution. Then, applying resolution to (1) and (2) we obtain $H(f(x)) \vee B(x)$ (4). Applying resolution to (4) and (3) and factoring the result we obtain $B(x)$. Now $B(x)$ can be used to simplify clauses (2) and (3). Therefore, on the instantiation side we can also simplify S into $\{\neg A(x) \vee H(x); B(x)\}$.

9 Saturation Strategies

Up to now we referred to the notion of a saturation process only informally. In this section we formalise this notion, and show that saturated sets can be achieved via fair saturation processes. First we define the notion of a saturation process for sets of ground closures. For a set of ground closures U, let $\overline{U}$ denote the set of clauses C such that $C \cdot \sigma$ is in U. An *Inst-Gen saturation process* is a sequence of triples, called *states*, $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^{\infty}$, where for every i, U^i is a set of ground closures, $I^i_\bot$ a model of $\overline{U}^i\bot$ and hsel^i a selection function based on that model. In addition we assume $U^1 \models \overline{U}^1$. Given a state $\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle$, a *successor state* $\langle U^{i+1}, I^{i+1}_\bot, \mathsf{hsel}^{i+1}\rangle$ is obtained by one of these steps:

– *(generation step)* $U^{i+1} = U^i \cup N$, where N is a set of ground closures such that $U^i \models N$; or
– *(elimination step)* $U^{i+1} = U^i \setminus N$, where every closure in N is redundant in U^i.

If for some i, $\overline{U}^i \bot$ is unsatisfiable the process terminates with the result "unsatisfiable". It immediately follows from the definition of a saturation process that in this case the initial set of clauses U^1 is unsatisfiable. Define $U^\cup = \cup_{i=1}^\infty U^i$. The set of *persistent closures* is defined as the low limit $U_\infty = \cup_{i\geq 1} \cap_{j\geq i} U^j$. We will use auxiliary lemmas about redundant sets of closures, these lemmas are similar to the corresponding lemmas in resolution setting [3]. For a set of ground closures U, let $\mathcal{R}(U)$ denote the set of all closures redundant in U.

Lemma 1. *Let U be a set of ground closures. Then, if a closure $C \cdot \sigma$ is redundant in U then $C \cdot \sigma$ is redundant in $U \setminus \mathcal{R}(U)$. In particular, $U \setminus \mathcal{R}(U) \models U$.*

Proof. Consider a closure $C \cdot \sigma \in \mathcal{R}(U)$. Let $M = \{C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n\}$ be the least subset of U, wrt. the multiset extension of $\succ$, such that $M \models C \cdot \sigma$. Then, all closures in M are non-redundant in U. Therefore, $M \subseteq U \setminus \mathcal{R}(U)$, and hence $C \cdot \sigma$ is redundant in $U \setminus \mathcal{R}(U)$. ❑

Lemma 2. *Let $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^\infty$ be a saturation process. Then, (i) $U^\cup \setminus \mathcal{R}(U^\cup) = U_\infty \setminus \mathcal{R}(U_\infty)$ and (ii) $\mathcal{R}(U^\cup) = \mathcal{R}(U_\infty)$.*

Proof. Let us prove (i).

($\subseteq$) If a ground closure $C \cdot \sigma \in U^\cup$ is not redundant in $U^\cup$ then $C \cdot \sigma$ is also not redundant in U^k for any k and therefore $C \cdot \sigma \in U_\infty$. Moreover, $C \cdot \sigma$ is not redundant in U_∞. Therefore $U^\cup \setminus \mathcal{R}(U^\cup) \subseteq U_\infty \setminus \mathcal{R}(U_\infty)$.

($\supseteq$) If a closure $C \cdot \sigma \in U_\infty$ is not redundant in U_∞, then by ($\subseteq$) direction, $C \cdot \sigma$ is not redundant in $U^\cup \setminus \mathcal{R}(U^\cup)$ and by Lemma 1 is not redundant in $U^\cup$. Therefore $U_\infty \setminus \mathcal{R}(U_\infty) \subseteq U^\cup \setminus \mathcal{R}(U^\cup)$.

Let us prove (ii). From Lemma 1 it follows that $\mathcal{R}(U^\cup) = \mathcal{R}(U^\cup \setminus \mathcal{R}(U^\cup))$ and similar $\mathcal{R}(U_\infty) = \mathcal{R}(U_\infty \setminus \mathcal{R}(U_\infty))$. Therefore, (i) implies $\mathcal{R}(U^\cup) = \mathcal{R}(U_\infty)$. ❑

First we note that a saturation process preserves (un)satisfiability of sets of clauses.

Lemma 3. *Let $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^\infty$ be a saturation process. Then, U^1 is satisfiable if and only if U_∞ is satisfiable.*

Proof. Implication from left to right follows trivially from the definition of a saturation process. In order to show implication from right to left, assume that U^1 is unsatisfiable. Then, $U^\cup$ is also unsatisfiable. Lemma 1 implies $U^\cup \setminus \mathcal{R}(U^\cup)$ is unsatisfiable. Since $U^\cup \setminus \mathcal{R}(U^\cup) \subseteq U_\infty$ we have U_∞ is unsatisfiable. ❑

In order to ensure that we obtain an Inst-Gen saturated set in the limit of the saturation process we need a notion of a fair saturation. For this we consider inference system SHInst-Gen on ground closures (see Section 5). Informally, a saturation process is fair if all non-redundant inferences between persisting closures are eventually applied or otherwise shown to be redundant. Let $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^\infty$ be a saturation process. An

SHInst-Gen inference between persistent closures $(\overline{L}'_1 \vee C_1) \cdot \sigma_1, \ldots, (\overline{L}'_k \vee C_k) \cdot \sigma_k$ and $(L_1 \vee \ldots \vee L_k \vee D) \cdot \sigma$ is called *SHInst-persistent* if there are an infinite number of indexes $j_1, \ldots j_i, \ldots$ such that these closures are in U^{j_i} and the inference is eligible (upon the same literals) at the state $\langle U^{j_i}, I^{j_i}_\bot, \mathsf{hsel}^{j_i} \rangle$ for all $i \geq 1$, that is, conditions (i)–(iv) on applicability of SHInst-Gen are satisfied. An Inst-Gen saturation process is *SHInst-fair* if every SHInst-Gen persisting inference in U_∞ is redundant wrt. $U^\cup$. Let us note that our redundancy criterion is *effective* in the sense of [3], that is adding the conclusion of the inference makes the inference redundant. Therefore, we can ensure fairness of a saturation process by adding conclusions of non-redundant SHInst-Gen persistent inferences.

Now we need to show that in the limit U_∞ of an SHInst-Gen fair saturation process we obtain a saturated set wrt. to a model of $U_\infty\bot$. If we compare our notion of saturation to saturation in the resolution framework (e.g., [3]), one of the key differences is that the literal selection can change at each step of the saturation. In particular, we need to construct a model $I^\infty_\bot$ of $U_\infty\bot$ and a selection hsel^∞ based on $I^\infty_\bot$ such that U_∞ is saturated wrt. hsel^∞. Although, compactness implies $U_\infty\bot$ is satisfiable if all $U^i\bot$ are satisfiable, not every model of $U_\infty\bot$ is suitable. Indeed, it is possible to construct an example of an SHInst-Gen fair saturation process with the limit U_∞ and a model I such that U_∞ is not saturated wrt. any selection function based on I. Another obstacle in constructing the required model $I^\infty_\bot$ is that a literal can be true in a model $I^i_\bot$ and its complement true in a model $I^j_\bot$ for $i \neq j$. Nevertheless, the following theorem shows that it is possible to construct a model $I^\infty_\bot$ of U_∞ and a selection function hsel^∞ based on $I^\infty_\bot$ such that U_∞ is saturated wrt. hsel^∞.

Lemma 4. [19] *Let U_∞ be a set of persistent clauses of a SHInst-Gen fair saturation process $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i \rangle\}^\infty_{i=1}$, and $U^i\bot$ is satisfiable for every i, $i \geq 1$. Then, there exists a model $I_\bot$ of $U_\infty\bot$ and a selection function* hsel *based on $I_\bot$ such that U_∞ is SHInst-Gen saturated wrt.* hsel.

Proof. Let $\{C_i \cdot \sigma_i\}^\infty_{i=1}$ be an enumeration of closures in U_∞. For each $n \geq 1$ we construct a partial interpretation J^n in which all $\{C_i\bot\}^n_{i=1}$ are true and a selection function hsel^n_J for $\{C_i\}^n_{i=1}$, based on J^n (meaning that all literals in $\mathsf{hsel}^n_J(C_i)\bot$ are true in J^n, i.e., true in all total consistent extensions of J^n, for $1 \leq i \leq n$) by induction on n. For each n the following invariants will be satisfied.

1. J^n is consistent and hsel^n_J is a selection function for clauses $\{C_i\}^n_{i=1}$ based on J^n.
2. $J^{n-1} \subseteq J^n$ and hsel^n_J coincides with hsel^{n-1}_J on clauses $\{C_i\}^{n-1}_{i=1}$.
3. There are infinitely many k such that for the model $I^k_\bot$ of $U^k\bot$ we have $J^n \subseteq I^k_\bot$ and for all $1 \leq i \leq n$, $\mathsf{hsel}^k(C_i) = \mathsf{hsel}^n_J(C_i)$.

If $n = 1$, then it is easy to see that there is a multiset M_1 of literals in $C_1\bot$ such that for infinitely many k, $\mathsf{hsel}^k(C_1) = M_1$. We take $J^1 = \bigcup_{L \in M_1}\{L\bot\}$ and $\mathsf{hsel}^1_J(C_1) = M_1$. It is immediate that all invariants (1–3) on J^1, hsel^1_J are satisfied.

Let $n \geq 1$ and assume that we have a model J^n and hsel^n_J for $\{C_i\}^n_{i=1}$ such that invariants (1–3) are satisfied. Since $(C_{n+1} \cdot \sigma_{n+1}) \in U_\infty$ we have that for some m and every $p \geq m$, $(C_{n+1} \cdot \sigma_{n+1}) \in U^p$. From this and invariant (3) it follows that for some $M_{n+1} \subseteq C_{n+1}$ there are infinitely many k such that: (i) $J^n \subseteq I^k_\bot$, and (ii)

$\mathsf{hsel}^k(C_i) = \mathsf{hsel}^n_J(C_i)$ for all $1 \leq i \leq n$, and (iii) $\mathsf{hsel}^k(C_{n+1}) = M_{n+1}$. Define $J^{n+1} = J^n \cup \bigcup_{L \in M_{n+1}} \{L\bot\}$ and $\mathsf{hsel}^{n+1}_J(C_i) = \mathsf{hsel}^n_J(C_i)$ for $1 \leq i \leq n$, and $\mathsf{hsel}^{n+1}_J(C_{n+1}) = M_{n+1}$. It is easy to see that all invariants (1–3) are satisfied for J^{n+1} and hsel^{n+1}_J.

We define $J = \cup_{i=1}^{\infty} J^i$ and $\mathsf{hsel}(C_i) = \mathsf{hsel}^i_J(C_i)$ for $i \geq 1$. From compactness and invariants (1) and (2), it follows that J is consistent, and hsel is a selection function based on J. We define $I_\bot$ as a total consistent extension of J, (note that hsel is also based on $I_\bot$).

Now we need to show that U_∞ is SHInst-Gen saturated wrt. hsel. Consider a SHInst-Gen inference from closures $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n$ in U_∞. Then, from the construction of hsel and in particular from the invariant (3) it follows that for infinitely many indexes k we have $\mathsf{hsel}^k(C_i) = \mathsf{hsel}(C_i)$, for all $1 \leq i \leq n$. Since the SHInst-Gen saturation process is fair, this inference is redundant in $U^\cup$ and by Lemma 2 is redundant in U_∞. Therefore, U_∞ is SHInst-Gen saturated wrt. hsel. ❑

We summarise the obtained results in the following theorem.

Theorem 7. *Let $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^{\infty}$, be a SHInst-Gen fair saturation process. Then, either:*

1. *for some i, $\overline{U}^i\bot$ is unsatisfiable and therefore U^1 is unsatisfiable, or*
2. *for every i, $\overline{U}^i\bot$ is satisfiable and therefore (by Lemmas (3,4) and Theorem 4) U^1 is satisfiable.*

Moreover, if for some i, U^i is SHInst-Gen saturated then at this stage we can conclude that U^1 is satisfiable.

In particular, Theorem 7 implies that if a set of closures U is unsatisfiable, then any SHInst-Gen fair saturation process $\{\langle U^i, I^i_\bot, \mathsf{hsel}^i\rangle\}_{i=1}^{\infty}$ with the initial set $U^1 = U$ terminates in a finite number of steps, proving unsatisfiability of U.

Saturation for sets of constrained clauses. In practice, we do not deal with closures directly, but rather with constrained clauses and inference systems such as DSInst-Gen (see Section 6). In this case a saturation process for clauses naturally corresponds to an Inst-Gen saturation process for closures. For a set of constrained clauses S, let $\widetilde{S}$ denote the set of all unconstrained clauses obtained from S by dropping all constraints. A *DSInst-Gen saturation process* is a sequence of triples $\{\langle S^i, I^i_\bot, \mathsf{sel}^i\rangle\}_{i=1}^{\infty}$, where S^1 is well-constrained, S^i is a set of constrained clauses such that $I^i_\bot$ a model of $\widetilde{S}^i\bot$ and sel^i a selection function based on $I^i_\bot$, for $i \geq 1$. Given $\langle S^i, I^i_\bot, \mathsf{sel}^i\rangle$, a *successor state* $\langle S^{i+1}, I^{i+1}_\bot, \mathsf{sel}^{i+1}\rangle$ is obtained by one of these steps:

- *(generation step)* $S^{i+1} = S^i \cup N$, where N is a set of constrained clauses such that $S^i \models Cl(N)$; or
- *(elimination step)* $S^{i+1} = S^i \setminus N$, where every constrained clause in N is redundant in S^i; or
- *(constraint extension step)* $S^{i+1} = (S^i \setminus \{C \mid [\,\varphi\,]\}) \cup \{C \mid [\,\varphi \wedge \psi\,]\}$, where $C \mid [\,\varphi\,]$ is in S^i and closures in $Cl(C \mid [\,\varphi\,]) \setminus Cl(C \mid [\,\varphi \wedge \psi\,])$ are redundant in $Cl(S^i)$.

Let us note that an inference by DSInst-Gen can be split into two saturation steps: generation and constraint extension steps. For simplicity, we assume that for every i and every clause $C \in \widetilde{S}^i$ there is only one constrained clause $C \mid [\,\varphi\,] \in S^i$. If there are several constrained clauses corresponding to the same unconstrained clause, we can replace them with one constrained clause by merging the constraints. Let $\widetilde{S}_\infty = \cup_{i\geq 1} \cap_{j\geq i} \widetilde{S}^j$. Consider two persistent clauses $C, D \in \widetilde{S}_\infty$. Let p be such that for all $q \geq p$ we have $C, D \in \widetilde{S}^q$, and corresponding constrained clauses are $C \mid [\,\varphi^q\,], D \mid [\,\psi^q\,] \in S^q$. We say that an DSInst-Gen inference associated with C and D is *DSInst-Gen persistent* if there is an infinite number of indexes $j \geq q$, such that the inference is eligible on clauses $C \mid [\,\varphi^j\,], D \mid [\,\psi^j\,] \in S^q$ at the stage j (upon the same selected literals), that is conditions (i)–(v) on applicability of DSInst-Gen are satisfied. A DSInst-Gen saturation process is *DSInst-Gen fair* if every DSInst-Gen persisting inference, associated with clauses in $\widetilde{S}_\infty$, is redundant in S^i for some i.

Theorem 7 can be applied to show completeness of fair DSInst-Gen saturation processes.

Theorem 8. *Let $SP = \{\langle S^i, I^i_\bot, \mathsf{sel}^i\rangle\}^\infty_{i=1}$, be a DSInst-Gen fair saturation process. Then, either:*

1. *for some i, $\widetilde{S}^i\bot$ is unsatisfiable and therefore S^1 is unsatisfiable, or*
2. *for every i, $\widetilde{S}^i\bot$ is satisfiable and therefore S^1 is satisfiable.*

Moreover, if for some i, S^i is DSInst-Gen saturated, then at this stage we can conclude that S^1 is satisfiable.

Proof. If for some i, $\widetilde{S}^i\bot$ is unsatisfiable then from the definition of DSInst-Gen saturation process it immediately follows that S^1 is unsatisfiable.

Let us assume that for every i, $\widetilde{S}^i\bot$ is satisfiable. With the DSInst-Gen saturation process SP we associate an Inst-Gen saturation process on ground closures $\widehat{\mathrm{SP}} = \{\langle \widehat{S}^i, I^i_\bot, \mathsf{sel}^i\rangle\}^\infty_{i=1}$, where $\widehat{S}^i = Cl(S^i)$. It is straightforward to check that if SP is DSInst-Gen fair then $\widehat{\mathrm{SP}}$ is Inst-Gen fair. Now, Theorem 7 implies that $\widehat{S}^1$ is satisfiable. Since S^1 is equivalent to $\widehat{S}^1$ we conclude that S^1 is also satisfiable. ❑

10 The effectively propositional fragment

Let us consider *the effectively propositional fragment (EPR)*, also called *the Bernays-Schönfinkel fragment*. The EPR is a clausal fragment of first-order logic where the signature is restricted to contain only predicate symbols and constants. This fragment is decidable and recently has been shown to have a number of applications ranging from hardware verification [17, 28, 40] to ontological reasoning [49], see [4] for more examples. Let us show that DSInst-Gen is a decision procedure for the EPR fragment.

We call a fair DSInst-Gen saturation process *pure* if all generation and constraint extension steps are results of application of DSInst-Gen inferences. For a set of constrained clauses S, let $\widehat{S}$ denote the set of ground closures represented by S. Consider a pure DSInst-Gen saturation process $\{\langle S^i, I^i_\bot, \mathsf{sel}^i\rangle\}^\infty_{i=1}$ with a finite set of initial clauses $S = S^1$. First, it is easy to see that any inference step is strictly reductive, that is if S' is

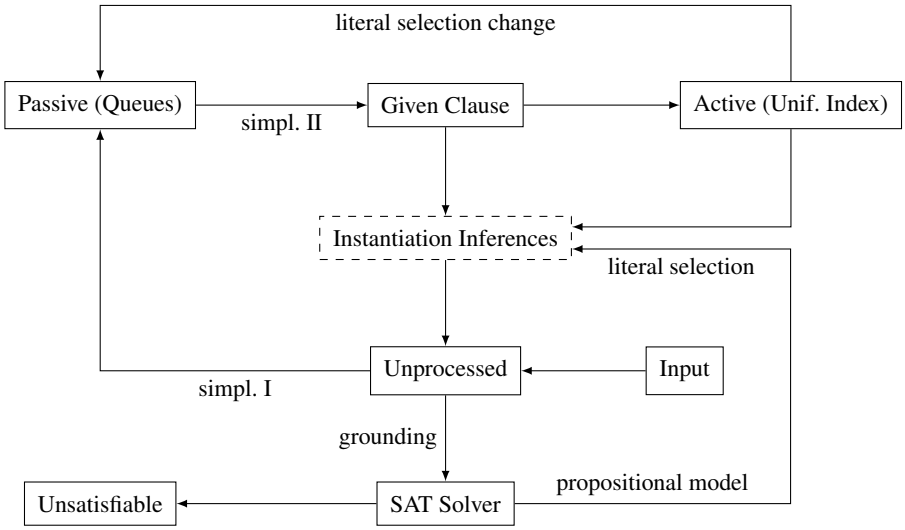

Fig. 1. Inst-Gen loop

obtained from S by application of a DSInst-Gen inference, then $\widehat{S} \succ_m \widehat{S}'$ (where $\succ_m$ is the multiset extension of $\succ$). Likewise, any elimination step is either strictly reductive or does not change the set of clauses. Since the initial set of clauses S is finite and all functional symbols in our signature are constants, the set $\widehat{S}$ is also finite. Therefore, after a finite number of steps n, the set of clauses will be stabilised, i.e., $S^n = S^{n+k}$ for $k \geq 0$. Then, Theorem 8 implies S is unsatisfiable if and only if $\widetilde{S}_n \bot$ is unsatisfiable. We summarise this in the following theorem.

Theorem 9. *DSInst-Gen is a decision procedure for the effectively propositional fragment.*

Experimental results presented in Section 12 show that instantiation-based methods and in particular DSInst-Gen are currently leading on the EPR problems.

11 Implementation of Inst-Gen in iProver

In previous sections we considered instantiation calculi, redundancy criteria and complete saturation strategies. Now we are ready to discuss implementation issues based on our implementation called iProver. iProver is a reasoning system for first-order logic based on the DSInst-Gen calculus. iProver incorporates a class of complete strategies and concrete redundancy elimination methods which we have discussed in the previous sections. At the core of iProver is the *Inst-Gen loop* (see Fig. 1) which governs the instantiation strategy.

Let us informally describe the Inst-Gen loop and its major components. The Inst-Gen loop is a modification of a well-known given clause algorithm which is a basis for all state-of-the-art resolution-based theorem provers. One of the main ideas of the

given clause algorithm is to separate clauses into two sets, called *active* and *passive* with the following properties. The set of active clauses is such that all non-redundant inferences between clauses in this set are performed. The set of passive clauses are the clauses waiting to participate in inferences. Let us consider the simplest version of the given clause algorithm for inference systems such as resolution, where for each clause, the set of literals eligible for inferences is fixed. Initially, the passive set consists of the input clauses and the active set is empty. The given clause algorithm consists of a loop and at each loop iteration the following actions are executed. First, a clause is taken from the passive set, called the *given clause*. Then, all inferences between the given clause and clauses in the active set are performed. Finally, all newly derived clauses are moved to passive and the given clause is moved to the active set. If the calculus is sound and at some stage the inconsistency is found, then the input set of clauses is inconsistent. If the calculus is complete and the selection strategy for the given clauses is fair, then the given clause algorithm will find an inconsistency in a finite number of steps. Moreover, if the calculus is complete and the given clause algorithm terminates with an empty passive set, then the active set of clauses is saturated and we can conclude that the input set of clauses is satisfiable. Redundancy elimination can be integrated into the given clause algorithm in various ways: passive and active clause sets are completely simplified at each iteration of the algorithm (the Otter loop [37]), or only active clauses are kept inter-simplified (the DISCOUNT loop [13]). In addition, preprocessing can be applied to the generated clauses such as splitting without backtracking (see [45]).

In order to adapt the standard given clause algorithm for instantiation strategies we need to: (i) accommodate propositional reasoning, and (ii) reflect dynamic literal selection (based on a propositional model of the ground abstraction), which can result in moving clauses from active to passive sets. On Fig. 1 we present such an adaptation of the given clause algorithm to the Inst-Gen framework: the Inst-Gen loop. Let us overview key components of the Inst-Gen loop and how they are implemented in iProver.

Passive. The passive set are the clauses waiting to participate in inferences. It is well-known that in the resolution-based setting, the order in which clauses are selected for inferences from the passive set is an important parameter. Usually, preference is given to clauses which are heuristically more promising to derive the contradiction, or to the clauses on which basic operations are easier to perform. In iProver, the passive clauses are represented by a sequence of priority queues. In order to define priorities we consider numerical/Boolean parameters of clauses such as: the number of symbols, the number of variables, the age of the clause, the number of literals, whether the clause is ground, the conjecture distance, whether the clause contains a symbol from the conjecture (other than equality or a theory symbol), whether the clause is Horn or in the EPR. Then, each queue is ordered by a lexicographic combination of orders defined on parameters. For example, if a user specifies an iProver option:

--inst_pass_queue1 [+age;-num_symb;+ground]

then in the first queue priority is given to clauses generated at the earlier iterations of the Inst-Gen loop (older clauses), then to clauses with fewer number of symbols and

finally to ground clauses. The user can also specify the ratio between the number of clauses taken from each queue.

Active. After the given clause is selected from the passive set all eligible inferences between the given clause and clauses in the active set should be performed. A unification index is used for efficient selection of clauses eligible for inferences. In particular, clauses in the active set are indexed by selected literals. The unification index implemented in iProver is based on non-perfect discrimination trees [21, 44]

Let us note that since the literal selection is based on a propositional model of the ground abstraction of the current set of clauses, selection can be changed during the Inst-Gen loop iterations. This can result in moves of clauses from active to passive sets, as shown in Fig. 1 (literal selection change). Changes in selection function and moves of clauses from active to passive sets result in a number of nontrivial technical issues such as ensuring fairness, and minimising the number of moves and repeated work, which are beyond the scope of this paper.

Instantiation Inferences. Instantiation inferences in iProver are based on the DSInst-Gen calculus. In particular, constrained clauses, dismatching constraint checking and model-based literal selections are implemented. Dismatching constraints are implemented using a discrimination-type index on atomic constraints to facilitate efficient satisfiability checking.

Redundancy elimination. The following redundancy eliminations are implemented: blocking non-proper instantiations, dismatching constraints, tautology elimination and global subsumption for both ground and non-ground clauses. The user can select whether to simplify all newly generated clauses (simpl. I in Fig. 1) or only the given clause (simpl. II in Fig. 1) or apply simplifications at both stages.

Grounding and SAT Solver. Newly derived clauses are grounded and added to the propositional solver. Although, in our theoretical considerations we used the designated constant $\bot$ for grounding, it is easy to see that all our arguments remain valid if we use any ground term in place of $\bot$. In particular, for grounding, iProver selects a constant with the greatest number of occurrences in the input set of clauses, other heuristics for selecting the term for grounding are also interesting to investigate. After grounding, clauses are added to the propositional solver. Currently, iProver integrates MiniSAT [15] for propositional reasoning.

Learning Restarts. It can happen that the Inst-Gen loop fails to terminate due to a poor choice of the literal selection on the initial set of clauses. Indeed, initially the propositional solver contains only few instances of the input clauses, and therefore selection based on the corresponding propositional model can be inadequate. Although the model and selection can be changed at the later iterations, by that time, the prover can consume most of the available resources. In order to overcome this, iProver implements restarts of the saturation process, keeping generated propositional clauses in the propositional solver. After each restart, the propositional solver will contain more instances of clauses, this can help to find a better literal selection. In addition, after each restart, global subsumption becomes more powerful.

Equality. iProver integrates equality by adding (internally) the necessary axioms of equality with an option of using Brand's transformation [9]. Our experiments show that even this naive approach of equality integration works reasonably well in the instantiation-based setting, most likely due to the semantic literal selection and absence of recombination of clauses with equality axioms. For more advanced treatment of equality based on combination of ordered unit superposition with Inst-Gen and the corresponding system iProver-Eq we refer to [32, 33].

Model representations. Consider a state in which the passive set is empty and the ground abstraction is satisfiable. Then by the completeness Theorem 8, the set of input clauses is satisfiable. Let us also assume that for each inference by DSInst-Gen between active clauses (including redundant inferences) the corresponding dismatching constraint is added to the premise according to the application of the DSInst-Gen inference rule. This can be easily achieved during or after saturation. In this case we can extract a model representation based on the selected literals in the active set of clauses and accumulated dismatching constraints. Since dismatching constraints can be naturally expressed in the language of the ground term algebra we can represent models using first-order definitions of predicates in the ground term algebra. A detailed treatment of model representations is beyond the scope of this paper, let us only mention that iProver supports several model outputs based on positive/negative predicate definitions in the ground term algebra.

Finite models. iProver has a finite model finding mode inspired by translation of finite model finding into the EPR fragment [5], see also [11]. Since instantiation-based methods are very efficient on the EPR fragment this approach is particularly promising. The method is complete for finite model finding: if there is a finite model of the given set of clauses, then such a model will be eventually found in a finite number of steps.

Combination with Resolution. In addition to the Inst-Gen loop, iProver implements a complete saturation algorithm for ordered resolution. In this paper, we will not discuss our implementation of resolution in detail. Let us only mention that the saturation algorithm is based on the same data structures as the Inst-Gen loop and implements a number of simplifications such as forward and backward subsumption, forward and backward subsumption resolution, tautology deletion and global subsumption. We implemented a compressed feature vector index (an extension of the feature vector index [48]) for efficient forward/backward subsumption and subsumption resolution.

Resolution is combined with instantiation by sharing the propositional solver. In particular, Ω-instances of clauses generated by resolution and instantiation are added to the propositional solver and propositional solver is used for global subsumption in both resolution and instantiation saturation loops.

12 Evaluation

In this section we evaluate iProver v0.9 on the standard benchmark for first-order theorem provers: the TPTP library [50] with the current version 5.2.0. Currently, iProver

does not have a built-in clausifier and we used Vampire [23] for clausification. iProver has also an interface for clausification using E prover [47] or any user provided clausifier which can output clauses in the TPTP format. Experiments were run on a cluster of Dell rack servers under Linux v2.6.30 with cpu 2.3GHz, memory 2GB and time limit 300s.

The TPTP library contains 15386 first-order problems, out of which 12420 are unsatisfiable, 1949 satisfiable and 1017 with unknown status. iProver in the default mode solves 8554 problems: 7524 unsatisfiable and 1030 satisfiable. Problems in the TPTP are rated from 0 to 1, where problems with the rating 0 are easy and problems with the rating 1 can not be solved by any automated reasoning system, including older versions of iProver, at the time of evaluating the corresponding version of the TPTP library. In the current version of the TPTP, iProver solved 24 problems with the rating 1, and 157 with rating ≥ 0.9. This indicates that there is a large number of problems in the TPTP that can only be solved by iProver. In the satisfiability mode, which features finite model finding, iProver can show satisfiability of 1301 problems out of 1949 known to be satisfiable in the TPTP.

It is interesting to compare an instantiation-based prover with an ordered resolution prover. iProver implements both on the same data structures and allows user to select combination of instantiation with resolution, pure instantiation and pure ordered resolution. Let us note that in iProver equality reasoning is integrated only in an axiomatic way in both instantiation and resolution parts, we refer to iProver-Eq [32] for a superposition-based integration. Results are presented in Table 1. We can see that instantiation considerably outperforms ordered resolution in this setting and the combination of instantiation and resolution leads to further improvements.

TPTP v5.2.0	Combination Inst. & Res.	Instantiation	Resolution
Solved	8554	7680	5724
Unsat.	7524	6731	5160
Sat.	1030	949	564

Table 1. iProver v0.9

We compare iProver v0.9 with other state-of-the-art automated reasoning systems based on the results of the CASC-23 competition, held in 2011 [51]. Tables are reproduced from the competition site[2], among different versions of the same system we take one with the best result. In the major FOF division, Table 2, iProver is in the top three provers along with established leaders Vampire [23, 46] and E [47].

In the EPR division, shown in Table 3, iProver considerably outperforms resolution/superposition based systems. The EPR fragment is of a particular interest since, as mentioned in the introduction, it has a wide range of applications.

Table 4 shows results in the first-order non-theorems (FNT) division. The FNT division corresponds to satisfiable first-order problems. Efficient methods for showing

[2] http://www.cs.miami.edu/~tptp/CASC/23/

FOF	Vampire	EP	iProver	leanCoP	iProver-Eq	EKRHyper	EDarwin	Metis	LEOII	Otter	Muscadet
300	0.6	1.4	0.9	2.2	0.7	1.2	1.4	2.3	1.2.8	3.3	4.1
Solved	269	232	192	136	135	109	103	101	97	62	42
av. time	12.95	22.55	9.22	46.80	8.68	8.93	6.97	24.75	25.18	5.84	8.99

Table 2. CASC-23 (FOF division, 300 problems)

EPR	iProver	Vampire	iProver-Eq	E	Metis	E-Darwin	FIMO	E-KHyper
150	0.9	1.8	0.7	1.4	2.3	1.4	0.2	1.2
Solved	145	127	121	91	78	70	62	60
av. time	12.70	15.79	24.78	7.90	20.85	12.64	1.81	10.02

Table 3. CASC-23 (EPR division, 150 problems)

satisfiability are usually based on finite model finding techniques. iProver capitalises on the translation of the finite model finding problem into the EPR fragment which helped to place iProver amongst top three system in this division.

To summarise, iProver performs well on both unsatisfiable and satisfiable problems over the whole TPTP and is leading in the EPR division.

FNT	Paradox	FIMO	iProver	Nitrox	iProver-Eq	EKRHyper	EP	EDarwin
200	3.0	0.2	0.9	0.2	0.7	1.2	1.4	1.4
Solved	169	162	159	140	86	85	78	57
av. time	3.33	14.43	34.93	17.42	7.52	15.92	2.40	7.23

Table 4. CASC-23 (FNT division: first-order non-theorems, 200 problems)

13 Conclusions

In this paper we have presented a development of the Inst-Gen framework from theoretical foundations to a working implementation. We considered the Inst-Gen calculus, semantic selection, hyper-inferences, redundancy elimination, dismatching constraints, simplifications by propositional reasoning, saturation strategies and finally implementation issues and evaluation. There are number of further extensions, that were not considered in this paper, such as integration of equational [19, 33] and theory reasoning in the black-box style [20]. These extensions open novel opportunities to utilise efficient solvers modulo theories, SMT solvers, which have recently gained great popularity due to demand in applications such as software and hardware verification.

Although our implementation is relatively new, iProver is amongst the leading systems and shows great potential of the Inst-Gen framework. We expect that integration of theory reasoning will greatly enhance applicability of iProver in domains such as verification of software and hardware.

To conclude, we believe that instantiation-based theorem proving, backed by theoretical foundations and state-of-the-art implementation techniques, is a promising approach which can be developed to be utilised in real-world applications.

14 Acknowledgements

I am very grateful to Harald Ganzinger who introduced me to the area of instantiation-based reasoning and with whom I had the great pleasure of investigating this area. I thank Andrei Voronkov for his encouragement, support and helpful suggestions. I thank Yevgeny Kazakov and an anonymous reviewer for many useful suggestions and Christoph Sticksel for providing the scripts used in the evaluation.

References

1. Behzad Akbarpour and Lawrence C. Paulson. Extending a resolution prover for inequalities on elementary functions. In N. Dershowitz and A. Voronkov, editors, *the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4790 of *LNCS*, pages 47–61. Springer, 2007.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University press, Cambridge, 1998.
3. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.
4. P. Baumgartner. Logical engineering with instance-based methods. In *the 21st International Conference on Automated Deduction*, volume 4603 of *LNCS*, pages 404–409. Springer, 2007.
5. P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *J. Applied Logic*, 7(1):58–74, 2009.
6. P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.
7. P. Baumgartner and R. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Third International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 125–139. Springer, 2006.
8. P. Baumgartner and C. Tinelli. The model evolution calculus. In *Proc. CADE-19*, number 2741 in LNAI, pages 350–364. Springer, 2003.
9. D. Brand. Proving theorems with the modification method. *SIAM J. Comput.*, 4(4):412–430, 1975.
10. R. Caferra and N. Zabel. A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13(6):613–641, 1992.
11. K. Claessen and N. Sorensson. New techniques that improve mace-style finite model finding. In P. Baumgartner and C. Fermueller, editors, *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications (MODEL'2003)*, 2003.
12. L. M. de Moura and N. Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. In *the 4th International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 410–425. Springer, 2008.
13. J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT - A distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

14. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT'2005*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.
15. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. of the 6th International Conference on Theory and Applications of Satisfiability Testing, SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.
16. T. Eiter, W. Faber, and P. Traxler. Testing strong equivalence of datalog programs - implementation and examples. In *the 8th International Conference LPNMR'05*, volume 3662 of *LNCS*, pages 437–441, 2005.
17. M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov. Encoding industrial hardware verification problems into effectively propositional logic. In R. Bloem and N. Sharygina, editors, *the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD'10)*, pages 137–144. IEEE, 2010.
18. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proc. 18th IEEE Symposium on LICS*, pages 55–64. IEEE, 2003.
19. H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *CSL'04*, volume 3210 of *LNCS*, pages 71–84, 2004.
20. H. Ganzinger and K. Korovin. Theory Instantiation. In *Proceedings of the 13 Conference on Logic for Programming Artificial Intelligence Reasoning (LPAR'06)*, volume 4246 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2006.
21. P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer, 1996.
22. M. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for CNF formulas. In C. G. Fermüller and A. Voronkov, editors, *the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *LNCS*, pages 357–371. Springer, 2010.
23. K. Hoder, L. Kovács, and A. Voronkov. Interpolation and symbol elimination in Vampire. In J. Giesl and R. Hähnle, editors, *the 5th International Joint Conference on Automated Reasoning, IJCAR'2010*, volume 6173 of *LNCS*, pages 188–195. Springer, 2010.
24. J.N. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first order logic. *Journal of Automated Reasoning*, 28:371–396, 2002.
25. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In *the Ninth International Conference on Principles of Knowledge Representation and Reasoning*, pages 152–162. AAAI Press, 2004.
26. U. Hustadt and R. Schmidt. MSPASS: modal reasoning by translation and first-order resolution. In *International Conference, TABLEAUX 2000*, volume 1847 of *Lecture Notes in Computer Science*, pages 67–71. Springer, 2000.
27. D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.
28. Z. Khasidashvili, M. Kinanah, and A. Voronkov. Verifying equivalence of memories using a first order logic theorem prover. In *the 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD'09*, pages 128–135. IEEE, 2009.
29. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Francaise d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automated deduction.
30. K. Korovin. iProver v0.2. In G. Sutcliffe, editor, *The CADE-21 ATP System Competition (CASC-21)*, 2007. see also [31].
31. K. Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In *the 4th International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

32. K. Korovin and C. Sticksel. iProver-Eq: an instantiation-based theorem prover with equality. In J. Giesl and R. Hähnle, editors, *the 5th International Joint Conference on Automated Reasoning, IJCAR'2010*, volume 6173 of *LNCS*, pages 196–202. Springer, 2010.
33. K. Korovin and C. Sticksel. Labelled unit superposition calculi for instantiation-based reasoning. In *the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2010)*, volume 6397 of *LNCS*, pages 459–473. Springer, 2010.
34. J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–317, 1987.
35. S.-J. Lee and D. Plaisted. Eliminating duplication with the Hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
36. C. Lynch and R. E. McGregor. Combining instance generation and resolution. In *7th International Symposium on Frontiers of Combining Systems, (FroCos'09)*, volume 5749 of *LNCS*, pages 304–318. Springer, 2009.
37. W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.
38. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 371–443. Elsevier, 2001.
39. J. A. Navarro Pérez. *Encoding and Solving Problems in Effectively Propositional Logic*. PhD thesis, University of Manchester, 2007.
40. J. A. Navarro Pérez and A. Voronkov. Encodings of bounded LTL model checking in effectively propositional logic. In *the 21st International Conference on Automated Deduction, (CADE'07)*, volume 4603 of *LNCS*, pages 346–361. Springer, 2007.
41. J. A. Navarro Pérez and A. Voronkov. Proof systems for effectively propositional logic. In *the 4th International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 426–440. Springer, 2008.
42. R. Pichler. Explicit versus implicit representations of subsets of the Herbrand universe. *Theor. Comput. Sci.*, 290(1):1021–1056, 2003.
43. D. Plaisted and Y. Zhu. Ordered semantic hyper-linking. *J. Autom. Reasoning*, 25(3):167–217, 2000.
44. I. V. Ramakrishnan, R. C. Sekar, and A. Voronkov. Term indexing. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier and MIT Press, 2001.
45. A. Riazanov and A. Voronkov. Splitting without backtracking. In *Proc. of the 17 International Joint Conference on Artificial Intelligence, (IJCAI'01)*, pages 611–617. Morgan Kaufmann, 2001.
46. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
47. S. Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.
48. S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proc. of the IJCAR-2004 Workshop on Empirically Successful First-Order Theorem Proving, Cork, Ireland*, ENTCS. Elsevier Science, 2004.
49. M. Suda, C. Weidenbach, and P. Wischnewski. On the saturation of YAGO. In J. Giesl and R. Hähnle, editors, *the 5th International Joint Conference on Automated Reasoning, IJCAR'2010*, volume 6173 of *LNCS*, pages 441–456. Springer, 2010.
50. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
51. G. Sutcliffe. CASC-23 proceedings of the CADE-23 ATP system competition, 2011. available at http://www.cs.miami.edu/~tptp/CASC/23/Proceedings.pdf.