An Invitation to Instantiation-Based Reasoning: From Theory to Practice

Konstantin Korovin*

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

Abstract. In this paper we survey results on an instantiation-based method for automated reasoning, called Inst-Gen, which was initiated by Harald Ganzinger and has been developed in [8–10]. Instantiation-based theorem-proving is a novel approach to automated reasoning which aims to utilise industrial-strength propositional SAT technology in the more general context of first-order logic. In our previous work we introduced a novel framework for proving completeness of various instantiation calculi, likewise presented concrete complete calculi for first-order logic with appropriate redundancy criteria. We showed how to integrate efficient equational reasoning and also theory reasoning in a black-box style. This theoretical work has provided a foundation for a new automated reasoning system, called iProver, which demonstrates the practicability of our instantiation-based method. In this paper we start by overviewing theoretical developments of an instance-based method Inst-Gen, describe its refinements, abstract redundancy elimination and conclude with concrete redundancy criteria and implementation issues.

1 Introduction

Most methods for automated first-order reasoning interleave a generation of suitable instances of clauses with applications of propositional-type rules to the obtained instances. Such 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. Examples from the first category include resolution calculus (see, e.g., [2]), disconnection calculus DCTP [18] and recent Model Evolution calculus ME [3].

Our approach to instantiation-based reasoning falls into the second category where propositional reasoning is integrated in a modular fashion and was motivated by earlier works (see [12, 17]). One of our main goals was 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 powerful reasoning systems which can be used in

* Supported by The Royal Society

practical applications. We show that most of the powerful machinery developed in the resolution-based framework (see [2, 20]), can be suitably adapted for our instantiation-based method.

In this paper we present a unifying view on the Inst-Gen framework. We start with abstract theoretical developments such as instantiation calculi, completeness results and abstract redundancy criteria. We then present concrete refinements such as semantic selection, dismatching constraints, simplifications utilising propositional reasoning and saturation strategies. Finally, we discuss implementation issues and evaluation.

In addition to an exposition of previous methods, developed with Harald Ganzinger [8–10], the paper also contains some novel results such as simplifications utilising propositional reasoning. These simplifications can be applied not only in the context of instantiation calculi but also in variety of other calculi, where strict subsumption is an admissible simplification, such as resolution. We hope that this paper will encourage the reader to investigate further developments and applications of the Inst-Gen framework, most notably, integration of equational [9] and theory reasoning [10].

2 Preliminaries

We adopt standard terminology used in first-order reasoning [2, 20]. 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 $\bot$.

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.

Substitutions are defined as usual and will be denoted by the letters ρ, σ, τ, and θ. 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. *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 also use $\bot$ to denote 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.

Instance-based theorem proving requires us to work 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 $C \cdot \sigma$. We work modulo renaming, that is, 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 be a clause in S, then a ground closure $C \cdot \sigma$ is called a *ground instance* of 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, $(A(x, y) \vee B(y)) \cdot [f(c)/x, c/y] \succ (A(f(u), v) \vee B(v)) \cdot [c/x, c/y]$. 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 called *true* (or *valid*) in a partial interpretation I, written $I \models C$, if C is true in each total extension of I, and is called *false* 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'. Truth values for closures are defined from the truth values of the clauses they represent.

3 Instantiation Calculus

The basic idea of instantiation-based reasoning is to reduce the unsatisfiability problem for sets of first-order clauses to a sequence of propositional unsatisfiability 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 $S\bot$ 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 can happen that S is unsatisfiable and we need to add more instances to S witnessing the unsatisfiability of S at the ground level. 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 instance generation we use instantiation calculus Inst-Gen and its refinements described below. Simplifications and redundancy elimination which are crucial for practical applicability of the method are described in Sections (5–7). We discuss strategies for interleaving application of inference rules, simplifications and propositional reasoning in Section 11.

We start with a basic calculus Inst-Gen and later add restrictions on this calculus and introduce redundancy criteria.

Inst-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 (mgu) of L and L',
(ii) θ is a proper instantiator of L or L'.

Let us informally describe the ideas behind Inst-Gen. Consider a set of clauses S and assume that the ground abstraction $S\bot$ of S is satisfiable. We can think of 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.

The soundness of Inst-Gen is obvious: conclusions of Inst-Gen logically follow from the premises. We can see that 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, applying resolution usually results in a rapid increase of clause lengths. Inst-Gen generates instances of the initial clauses and therefore the length of clauses does not increase. Likewise there is no recombination of clauses which can result in repeated work in the resolution setting (see [17] and Example 5).

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. [8] *Let S be a set of clauses such that $S\bot$ is satisfiable. If S is Inst-Gen saturated then S 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.

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 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. Suppose that $S = \{A(f(x)) \vee C(x); \neg A(y) \vee D(y)\}$ and $I_\bot$ is 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 conflicting instances of clauses 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, instance generation calculus SInst-Gen based on sel, is defined as follows.

SInst-Gen

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

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:

$$A(x_1, x_2) \ \vee \ A(x_2, x_3) \ \vee \ldots \vee \ A(x_{n-1}, x_n) \qquad (1)$$
$$\neg A(c_1, c_2) \vee \neg A(c_2, c_3) \vee \ldots \vee \neg A(c_{n-1}, c_n). \qquad (2)$$

Unrestricted applications of Inst-Gen will generate exponentially many (wrt. n) different instances of the first clause. Consider now SInst-Gen based on a selection function that selects the first literal from each of the clauses $\{(1), (2)\}$. We obtain the conclusion

$$A(c_1, c_2) \vee A(c_2, x_3) \vee \ldots \vee A(x_{n-1}, x_n). \qquad (3)$$

Now, selecting the second literal 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 do not terminate on such 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. Let $I_\bot$ be 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 $C \in S$ 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 $C \in S$. 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

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

where (i) θ is the most general unifier of $(L_1, L'_1), \ \ldots, \ (L_k, L'_k)$,
(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), $1 \leq i \leq k$. Now we can formulate the completeness theorem for SHInst-Gen, which also applies to SInst-Gen as a special case.

Theorem 2. [8] *Let S be a set of clauses such that $S\bot$ is satisfiable. If S is saturated by SHInst-Gen for 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 resolution calculus.

Let S be a set of clauses. A ground closure $C \cdot \sigma$ is called *redundant* in S if there exist ground closures $C_1 \cdot \sigma_1, \ldots, C_k \cdot \sigma_k$ that are ground instances of S 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 $1 \leq i \leq k$. In other words, a ground closure is redundant in S if it logically follows from smaller closures (w.r.t. $\succ$) in S. A clause C (possibly non-ground) is called redundant in S if each ground closure $C \cdot \sigma$ is redundant in S. This abstract redundancy criteria can be used to justify many standard redundancies. First, we note that tautologies are implied by any set of closures and therefore are redundant. Second, assume that $\succ$ satisfies the following natural property on ground clauses: (i) if C is a strict sub-multiset of C' (denoted $C \subset C'$) then $C' \succ C$. We say that C *strictly subsumes* C' if there is a substitution θ such that $C\theta \subset C'$. For example, $A(x, y)$ strictly subsumes $A(x, a) \vee B(x)$. It is easy to see that if the closure ordering $\succ$ satisfies (i), then strict subsumption is an admissible redundancy. We can refine strict subsumption further if we assume that $\succ$ in addition to (i) also satisfies the following: (ii) if $C\theta = C'$ and θ is not a proper instantiator and is not a renaming then $C' \cdot \sigma \succ C \cdot \theta\sigma$ for every grounding substitution σ. For example, if the closure ordering $\succ$ satisfies (ii), then $A(x, x) \vee B(x)$ is redundant w.r.t. $A(x, y) \vee B(y)$. We call closure orderings satisfying (i) and (ii) *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. 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 any 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:

$$\begin{array}{llll} \underline{A(x,y)} \vee B(y) & (1) & \underline{B(a)} & (2) \\ \neg\underline{A(a,z)} \vee C(z) & (3) & \underline{C(b)} & (4) \end{array}$$

Let us show that 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 (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 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.*

We can have additional flexibility with partial instantiations as follows. Let $C\theta$ be as in Proposition 2 and θ' be 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 $D = \underline{\neg A(f(a), b)}$. The most general unifier of $A(x, y)$ and $A(f(a), b)$ 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 [8], 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 *saturated up to redundancy* wrt. SHInst-Gen if all inferences in SHInst-Gen from premises in S are redundant in S. Next theorem shows that completeness is preserved under redundancy elimination.

Theorem 3. [8] *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 saturated up to redundancy under SHInst-Gen 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 finer control over redundancy we consider the notion of inferences and saturation at the level of ground closures. Let U be a set of ground closures, then $\overline{U}$ denotes 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)

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

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

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 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-saturated up to redundancy wrt.* sel *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 the model-generation technique (see [2, 20]).

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 we show that if U is SInst-saturated up to redundancy, then any total extension of I_U is indeed a model of U. Let $I_\bot$ be a model of $\overline{U}\bot$ and sel be a selection function on clauses in $\overline{U}$ based on $I_\bot$.

We construct I_U by induction on closures ordered by $\succ$. Informally, we construct our model by adding literals in a way to satisfy closures in U. Let $\widehat{C} = C \cdot \sigma$ be a ground closure in U. 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. $C\sigma$ is false in $I_{\widehat{C}}$ (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) $\widehat{C}$ is false in $I_{\widehat{C}}$, and (ii) $\widehat{C}$ is true in $I_{\widehat{C}} \cup \epsilon_{\widehat{C}}$ and hence $\widehat{C}$ is 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 the theorem follows. Assume otherwise. Let $\widehat{C} = C \cdot \sigma$ be the minimal ground closure in U that is false in $I_{\widehat{C}} \cup \epsilon_{\widehat{C}}$. Let $L = \mathsf{sel}(C)$. As $\widehat{C}$ is not productive and false in $I_{\widehat{C}}$ 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 $D\theta\tau' = D\tau$. Let us prove that this inference is not redundant. Indeed, assume that $C \cdot \theta\sigma' = C \cdot \sigma = \widehat{C}$ 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 to 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 $\widehat{D}$.

We have shown that there exists a non-redundant inference with premises in U, contradicting the assumption of the theorem that U is SInst-saturated up to redundancy. This concludes the proof of the 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 SInst-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 S. We have that $\overline{U} = S$ and U is SInst-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, and finally we show how 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 $C \cdot \sigma$ which are represented by $C\theta$ are redundant. 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. [5, 15, 20]). Dismatching constraints are particularly attractive: on the one hand they provide powerful restrictions for instantiation calculus, and on the other, checking dismatching constraints can be efficiently implemented.

A *simple dismatching constraint* is a formula $ds(\bar{s}, \bar{t})$, also denoted as $\bar{s} \lhd_{ds} \bar{t}$, where $\bar{s}$, $\bar{t}$ are two variable disjoint tuples of terms, with the following semantics. A solution to a constraint $ds(\bar{s},\bar{t})$ is a substitution σ such that for every substitution γ, $\bar{s}\sigma \not\equiv \bar{t}\gamma$. For example, consider a simple dismatching constraint $\varphi(x,y) = \langle x\rangle \lhd_{ds} \langle f(y)\rangle$. 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 a simple dismatching constraint $ds(\bar{s},\bar{t})$ is satisfiable if and only if for all substitutions γ, $\bar{s} \not\equiv \bar{t}\gamma$. In other words, a dismatching constraint $ds(\bar{s},\bar{t})$ is not satisfiable if and only if there is a substitution μ such that $\bar{s} \equiv \bar{t}\mu$, which is a familiar matching problem.

We will use conjunctions of simple dismatching constraints, called *dismatching constraints*, $\wedge_{i=1}^{n} ds(\bar{s}_i,\bar{t}_i)$, where $\bar{t}_i$ 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 $\wedge_{i=1}^{n} ds(\bar{s}_i,\bar{t}_i)$ if σ is a solution of each $ds(\bar{s}_i,\bar{t}_i)$, 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 matching problem which can be solved in linear-time (see, e.g., [1]).

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 $C \mid [\,\wedge_{i=1}^{n} ds(\bar{s}_i,\bar{t}_i)\,]$, 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 all ground closures $C \cdot \sigma$, denoted as $Cl(C \mid [\,\varphi\,])$, such that σ is a solution to φ. An unconstrained clause C can be seen as a constrained clause with an empty constraint $C \mid [\;]$. For a set S of constrained clauses, $Cl(S)$ denotes the set of all ground closures represented by constrained clauses from 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 $S \models \widetilde{S}$. 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

$$\frac{L \vee C \mid [\,\varphi\,] \quad \overline{L'} \vee D \mid [\,\psi\,]}{L \vee C \mid [\,\varphi \wedge \bar{x} \lhd_{ds} \bar{x}\theta\,] \quad (L \vee C)\theta}$$

where (i) 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
(ii) $\mathsf{sel}(L \vee C) = L$ and $\mathsf{sel}(\overline{L'} \vee D) = \overline{L'}$, and
(iii) θ is a proper instantiator for L, and
(iv) $\bar{x}$ is a tuple of all variables in 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 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.

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

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

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

$$\begin{array}{rl} \underline{\neg A(x)} \vee C(x) \mid [\, x \lhd_{ds} f(z) \,] & (0'), \\ \neg A(f(x)) \vee \underline{C(f(x))} & (0''), \end{array} \qquad \begin{array}{rl} \underline{A(f(y))} \vee D_1 & (1), \\ \underline{A(f^{i_2}(y))} \vee D_2 & (2), \\ \dots & \\ \underline{A(f^{i_n}(y))} \vee D_n & (n). \end{array}$$

We can see S' is saturated wrt. DSInst-Gen. 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 by each resolution inference. This can result in repeated work on the same instances of $C(x)$. Dismatching constraints allow one to avoid such problems in the instantiation setting.

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 $C \mid [\, \varphi \wedge \bar{x} \lhd_{ds} \bar{x}\theta \,]$, $C\theta$ 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 $Cl(S)$.

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 $x \lhd_{ds} f(y) \vee x \lhd_{ds} a$ 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 [14, 16, 22]. In contrast to the problem of satisfiability, which can be solved in linear-time, the problem of ground satisfiability is NP-complete.

Theorem 5. [14, 16] *The ground satisfiability problem for dismatching constraints is NP-complete.*

7 Simplification by Propositional Reasoning

Having at hand a powerful propositional solver it seems promising to develop methods for redundancy elimination which utilise propositional reasoning. Let us discuss some methods for redundancy elimination based on propositional 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 $Sim = \{D_1 \cdot \tau_1, \dots D_n \cdot \tau_n\}$ (not necessarily contained in U) such that:

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

We call Sim as 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 $C \cdot \sigma$. We call $D \cdot \tau$ a *strict subclosure* of $C \cdot \sigma$ (denoted by $D \cdot \tau \subset C \cdot \sigma$) if $D \subset C$ and $D\tau \subset C\sigma$. 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 8 we consider simplification of ground clauses, and in Section 9 simplification of clauses with variables.

8 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 $D \subset C$. 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 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 $D \subset C$ 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 $D \subset C$ 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 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}.

9 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 $\bar{c}$ be a tuple of pairwise different constants not in Σ. Then, $\models \varphi(\bar{c}) \rightarrow \psi(\bar{c})$ implies $\forall \bar{x} \varphi(\bar{x}) \models \forall \bar{x} \psi(\bar{x})$.*

Proof. We have the following implications: $\models \varphi(\bar{c}) \rightarrow \psi(\bar{c})$ equivalent to $\models \forall \bar{x}(\varphi(\bar{x}) \rightarrow \psi(\bar{x}))$ implies $\models (\forall \bar{x} \varphi(\bar{x})) \rightarrow (\forall \bar{x} \psi(\bar{x}))$ equivalent to $\forall \bar{x} \varphi(\bar{x}) \models \forall \bar{x} \psi(\bar{x})$.

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 $\gamma \in \Omega$. Let us assume that for any clause $C \in S$ there are some Ω-instances of C in 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 $S_{gr} \models D\gamma$ for some $\gamma \in \Omega$, and D' is not empty.

There are several non-trivial issues to consider when we try to apply global subsumption. The main issues are: (i) which Ω-instances of clauses in S to add to S_{gr}, (ii) which clause D to use as a candidate for the conclusion, and (iii) 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 (i) above we assume that for a clause $C \in S$, $C\gamma_C$ is in S_{gr}. For (ii), we choose candidates for the conclusion of global subsumption among strict subclauses of a given clause. For (iii), 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 [28]. 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 [13]. 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 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 pure propositional reasoning suffices for these simplifications. In practice, we can employ efficient propositional solvers for such simplifications in a black-box fashion. 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.

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 can be suitably used for such implication generation. In particular, most propositional solvers generate so called learnt clauses which are implied clauses and therefore learnt clauses can be used for simplifications of clauses in S.

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

10 Combination of Instantiation with Resolution

One of the attractive properties of instantiation calculus is that the length of clauses does not increase during instantiation. On the other hand, if we consider instantiation calculus without simplifications, the length of clauses does not decrease either. 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 (ordered) resolution calculus. There are different ways to combine instantiation with resolution, (see, e.g., [10]), 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 $\{\neg A(x) \vee H(x); B(x)\}$.

11 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-saturation process* is a sequence of triples $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i \rangle\}_{i=1}^{\infty}$, where U^i is a set of ground closures, $I^i_\perp$ a model of $\overline{U}^i\perp$ and hsel^i a selection function based on that model. In addition we assume $U^1 \models \overline{U}^1$. Given $\langle U^i, I^i_\perp, \mathsf{hsel}^i \rangle$, a *successor state* $\langle U^{i+1}, I^{i+1}_\perp, \mathsf{hsel}^{i+1} \rangle$ 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, $\overline{U}^i\perp$ 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 [2]. 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 $\{C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n\}$ be the minimal subset of U, wrt. the multiset extension of $\succ$, such that $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n \models C \cdot \sigma$. Then, all closures $C_i \cdot \sigma_i$ are non-redundant in U for $1 \leq i \leq n$. Therefore, $C \cdot \sigma$ is redundant in $U \setminus \mathcal{R}(U)$.

Lemma 2. *Let $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i \rangle\}_{i=1}^{\infty}$ 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 a saturation process preserves (un)satisfiability of sets of clauses.

Lemma 3. *Let $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i \rangle\}_{i=1}^{\infty}$ 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-saturated set in the limit of the saturation process we need the notion of Inst-fair saturation. For this we consider inference system SHInst-Gen on ground closures (see Section 5). Informally, a saturation process is Inst-fair if all non-redundant inferences between persisting closures are eventually applied or otherwise shown to be redundant. Let $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i \rangle\}_{i=1}^{\infty}$ 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 is an infinite number of indexes j such that the inference is eligible at the stage j of the saturation process (upon the same literals), that is, conditions (i)–(iv) on applicability of SHInst-Gen are satisfied. An Inst-Gen saturation process is *SHInst-fair* if every SHInst-persisting inference in U_∞ is redundant wrt. $U^\cup$. Let us note that our redundancy criterion is *effective* in the sense of [2], 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-persistent inferences.

Now we need to show that in the limit U_∞ of an SHInst-fair saturation process we obtain a saturated set wrt. to a model of $U_\infty\perp$. If we compare our notion of saturation to saturation in the resolution framework (e.g., [2]), 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 $I^\infty_\perp$ of $U_\infty\bot$ and a selection hsel^∞ based on $I^\infty_\perp$ 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-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 $I^\infty_\perp$ is that models $I^i_\perp$ and $I^j_\perp$ can be incompatible for $i \neq j$. Nevertheless, the following theorem shows that it is possible to construct a model $I^\infty_\perp$ of U_∞ and a selection function hsel^∞ based on $I^\infty_\perp$ such that U_∞ is saturated wrt. hsel^∞.

Lemma 4. [9] *Let U_∞ be a set of persistent clauses of a SHInst-fair saturation process $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i\rangle\}_{i=1}^\infty$, and $U^i_\infty\bot$ is satisfiable for every i, $i \geq 1$. Then, there exists a model $I_\perp$ of $U_\infty\bot$ and a selection function* hsel *based on $I_\perp$ such that U_∞ is SHInst-saturated wrt.* hsel.

Proof. Let $\{C_i \cdot \sigma_i\}_{i=1}^\infty$ be an enumeration of closures in U_∞. For each $n \geq 1$ we construct a partial interpretation J^n in which all $\{C_i\bot\}_{i=1}^n$ are true and a selection function hsel^n_J for $\{C_i\}_{i=1}^n$, 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 $\{C_i\}_{i=1}^n$ based on J^n.
2. $J^{n-1} \subseteq J^n$ and hsel^n_J coincides with hsel^{n-1}_J on clauses $\{C_i\}_{i=1}^{n-1}$.
3. There are infinitely many k such that for the model $I^k_\perp$ of $U^k\bot$ we have $J^n \subseteq I^k_\perp$ and for all $1 \leq l \leq n$, $\mathsf{hsel}^k(C_l) = \mathsf{hsel}^n_J(C_l)$.

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 $\{C_i\}_{i=1}^n$ 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) $J^n \subseteq I^k_\perp$, and (ii) $\mathsf{hsel}^k(C_l) = \mathsf{hsel}^n_J(C_l)$ for all $1 \leq l \leq n$, 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 $\mathsf{hsel}^{n+1}_J(C_l) = \mathsf{hsel}^n_J(C_l)$ for $1 \leq l \leq n$, 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 $I_\perp$ as a total consistent extension of J, (note that hsel is also based on $I_\perp$).

Now we need to show that U_∞ is SHInst-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-saturation process is fair, this inference is redundant in $U^{\cup}$ and by Lemma 2 is redundant in U_∞. Therefore, U_∞ is SHInst-saturated wrt. hsel.

We summarise the obtained results in the following theorem.

Theorem 6. *Let $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i\rangle\}_{i=1}^{\infty}$, be a SHInst-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-saturated then at this stage we can conclude that U^1 is satisfiable.

In particular, Theorem 6 implies that if a set of closures U is unsatisfiable, then any SHInst-fair saturation process $\{\langle U^i, I^i_\perp, \mathsf{hsel}^i\rangle\}_{i=1}^{\infty}$ 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-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-saturation process* is a sequence of triples $\{\langle S^i, I^i_\perp, \mathsf{sel}^i\rangle\}_{i=1}^{\infty}$, where S^i is a set of constrained clauses such that S^1 is well-constrained, $I^i_\perp$ a model of $\widetilde{S}\bot$ and sel^i a selection function based on that model. Given $\langle S^i, I^i_\perp, \mathsf{sel}^i\rangle$, a *successor state* $\langle S^{i+1}, I^{i+1}_\perp, \mathsf{sel}^{i+1}\rangle$ 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-inference associated with C and D is *DSInst-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-saturation process is *DSInst-fair* if every DSInst-persisting inference, associated with clauses in $\widetilde{S}_\infty$, is redundant in S^i for some i.

Theorem 6 can be applied to show completeness of fair DSInst-saturation processes.

Theorem 7. *Let $SP = \{\langle S^i, I^i_\perp, \mathsf{sel}^i\rangle\}_{i=1}^{\infty}$, be a DSInst-fair saturation process. Then, either:*

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

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

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

Let us assume that for every i, $\widetilde{S}^i\perp$ is satisfiable. With the DSInst-saturation process SP we associate an Inst-saturation process on ground closures $\widehat{\mathrm{SP}} = \{\langle \widehat{S}^i, I^i_\perp, \mathsf{sel}^i\rangle\}_{i=1}^{\infty}$, where $\widehat{S}^i = Cl(S^i)$. It is straightforward to check that if SP is DSInst-fair then $\widehat{\mathrm{SP}}$ is Inst-fair. Now, Theorem 6 implies that $\widehat{S}^1$ is satisfiable. Since S^1 is equivalent to $\widehat{S}^1$ we conclude that S^1 is also satisfiable.

Example 8. Let us consider the Bernays-Schönfinkel class where the signature is restricted to contain only predicate symbols and constants. We call a fair DSInst-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-saturation process $\{\langle S^i, I^i_\perp, \mathsf{sel}^i\rangle\}_{i=1}^{\infty}$ 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 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 7 implies S is unsatisfiable if and only if $\widetilde{S}_n\perp$ is unsatisfiable.

12 Implementation of iProver

In previous sections we have developed 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 DSInst-Gen calculus which implements a class of complete strategies and concrete redundancy elimination methods which were discussed in the previous sections. At the core of iProver is *Inst-Gen Loop* (see Fig. 1) which governs the instantiation strategy.

Let us informally describe Inst-Gen Loop and its major components. 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

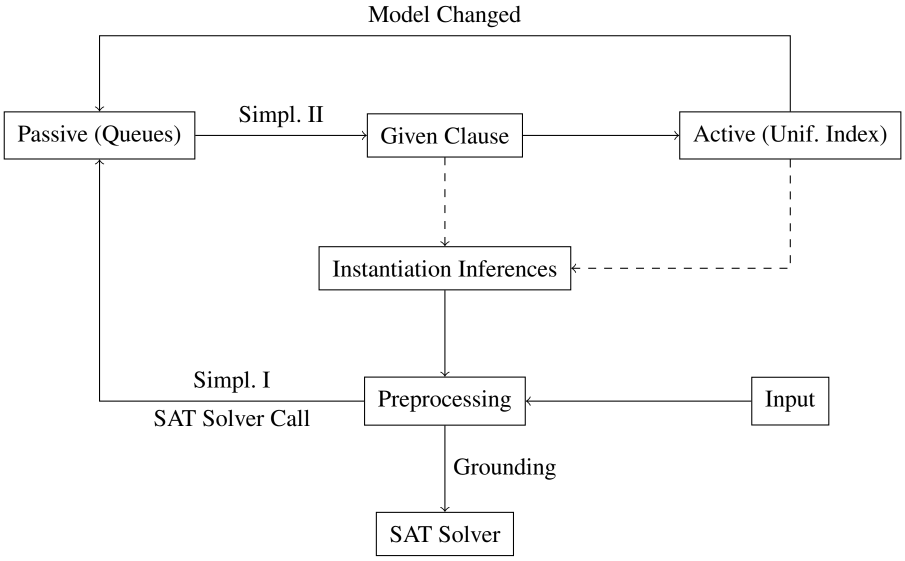

Fig. 1. Inst-Gen Loop

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 Active 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 Given Clause. Then, all inferences between Given Clause and clauses in Active are performed. Finally, all newly derived clauses are moved to Passive and Given Clause is moved to Active. 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 Given Clauses is fair, then the given clause algorithm will find inconsistency in a finite number of steps. Moreover, if the calculus is complete and the given clause algorithm terminates with empty Passive, then Active is a saturated set of clauses 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 are completely simplified at each iteration of the algorithm (Otter Loop [19]), or only Active clauses are kept inter-simplified (DISCOUNT Loop [6]). In addition, preprocessing can be applied to the generated clauses such as splitting without backtracking (see [23]).

As shown in Fig 1, Inst-Gen Loop is a modification of the standard given clause algorithm which accommodates propositional reasoning. Let us overview key components of Inst-Gen Loop and how they are implemented in iProver.

Passive. The Passive set are the clauses waiting to participate in inferences. From the experience with resolution-based systems it is known that the order in which clauses are selected for inferences from Passive 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, Passive clauses are represented by two priority queues. In order to define priorities we consider numerical/boolean parameters of clauses such as: number of symbols, number of variables, age of the clause, number of literals, whether the clause is ground, conjecture distance, whether the clause contains a symbol from the conjecture (other than equality or a theory symbol). 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 priority in the first queue is given to clauses generated at the earlier iterations of Inst 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 the first queue and the second queue.

Active. After Given Clause is selected from Passive all eligible inferences between Given Clause and clauses in Active should be performed. A unification index is used for efficient selection of clauses eligible for inferences. In particular, Active clauses are indexed by selected literals. Let us note that since the literal selection is based on a propositional model (of a 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, as shown in Fig 1. Changes in selection function and moves from Active to Passive result in a number of nontrivial technical issues such as ensuring fairness, and minimising the number of moves and repeated work. Since these issues are rather technical we will not consider them here. Let us only mention that the unification index implemented in iProver is based on non-perfect discrimination trees [11].

Instantiation Inferences. iProver implements DSInst-Gen calculus. In particular, constrained clauses, dismatching constraint checking and semantic-based literal selections are implemented.

Redundancy elimination. In addition to dismatching constraints, global subsumption for clauses with variables and tautology elimination are implemented.

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 $\perp$ for grounding, it is easy to see that all our arguments remain valid if we use any ground term in place of $\perp$. In particular, for grounding, iProver selects a constant with the greatest number of occurrences in the input set of clauses. After grounding, clauses are added to the propositional solver. Currently, iProver integrates MiniSat [7] solver for propositional reasoning.

Learning Restarts. It can happen that Inst-Gen Loop fails to find the solution 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. 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.

Combination with Resolution. In addition to Inst-Gen Loop, iProver implements a complete saturation algorithm for ordered resolution. In this paper, we will not discuss our implementation of resolution in details. Let us only mention that the saturation algorithm is based on the same data structures as Inst-Gen Loop and implements a number of simplifications such as forward and backward subsumption (based on a vector index [26]), subsumption resolution, tautology deletion and global subsumption. Resolution is combined with instantiation by sharing the propositional solver. In particular, groundings of clauses generated by resolution are added to the propositional solver and propositional solver is used for global subsumption in both resolution and instantiation saturation loops.

13 Evaluation

iProver v0.3 was evaluated on the standard benchmark for first-order theorem provers – TPTP library version v3.2.0[1]. Currently, iProver does not have a built-in clausifier and we used E prover[2] for clausification. Experiments were run on a cluster of PC's with CPU: 1.8GHz, Memory: 512 Mb, Time Limit: 300s, OS: Linux v2.6.22. Out of 8984 problems in the TPTP library, iProver (single strategy) solved 4814 problems: 3972 unsatisfiable and 842 satisfiable. Problems in TPTP are rated from 0 to 1, where problems with the rating 0 are easy and problems with the rating 1 can not not be solved by any automated reasoning system, (at the time of evaluating corresponding version of the TPTP library). iProver solved six problems with the rating 1, and 27 with rating greater than 0.9, (TPTP v3.2.0).

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 currently equality reasoning is integrated only in an axiomatic way in both instantiation and resolution parts. Results are presented in Table 1. We can see that instantiation considerably outperforms ordered resolution in this case and the combination of instantiation and resolution leads to further improvements.

We compare iProver v0.2 with other state-of-the-art automated reasoning systems based on results of the CASC-21 competition, held in 2007 [27] (tables are reproduced

[1] http://www.cs.miami.edu/˜tptp/
[2] www.eprover.org/

TPTP v3.2.0	Combination Inst. & Res.	Instantiation	Resolution
Solved	4814	4499	2859
Unsat.	3972	3692	2425
Sat.	842	807	434

Table 1. iProver v0.3

from the competition site[3]). In the major FOF devision, Table 2, iProver is in the top three provers along with established leaders Vampire [24] and E [25].

FOF	Vampire 9.0	E 0.999	iProver 0.2	Equinox 1.2	leanCoP 2.0	Otter 3.3	Metis 2.0	Geo 2007f	Muscadet 2.7
Attempted	300	300	300	300	300	300	300	300	300
Solved	270	248	201	173	160	138	117	104	37
Av. Time (s)	20.84	13.36	9.57	14.92	39.50	25.04	16.06	29.42	0.76

Table 2. CASC-21 (FOF Division)

In the EPR division, shown in Table 3, (which corresponds to Bernays-Schönfinkel class, see Section 11) iProver is on a par with Darwin, which is also an instantiation-based prover [4] (based on different ideas from this paper). This fragment of first-order logic is of a particular interest. On the one hand, most of instantiation-based approaches result in a decision procedure for Bernays-Schönfinkel class. On the other hand, this class has a number of important applications in the areas such as bounded model checking and planning [21].

EPR	Darwin 1.3	iProver 0.2	Paradox 2.2	E 0.999	Geo 2007f	Vampire 9.0	E-KRHyper 1.0	Metis 2.0	Equinox 1.2
Attempted	100	100	100	100	100	100	100	100	100
Solved	98	97	88	80	76	75	74	73	46
Av. Time (s)	1.87	4.66	2.69	4.31	2.90	2.10	1.79	8.68	13.07

Table 3. CASC-21 (FOF EPR)

14 Conclusions

In this paper we have presented a development of the Inst-Gen framework from theoretical foundations to a working implementation. We considered Inst-Gen calculus, semantic selection, hyper-inferences, redundancy elimination, dismatching constraints,

[3] http://www.cs.miami.edu/~tptp/CASC/21/

simplifications by propositional reasoning, saturation strategies and finally implementation issues and evaluation. There are number of further extensions, yet to be implemented, such as integration of equational [9] and theory reasoning in the black-box style [10]. 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 young, iProver is among the leading systems and shows great potential of the Inst-Gen framework. We expect that integration of equational and 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.

15 Acknowledgements

Most of the results presented in this paper were obtained in collaboration with Harald Ganzinger. I am grateful to Andrei Voronkov for his encouragement, support and helpful suggestions.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University press, Cambridge, 1998.
2. 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.
3. P. Baumgartner and C. Tinelli. The model evolution calculus. In *Proc. CADE-19*, number 2741 in LNAI, pages 350–364. Springer, 2003.
4. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.
5. 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.
6. J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT - A distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.
7. 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.
8. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proc. 18th IEEE Symposium on LICS*, pages 55–64. IEEE, 2003.
9. H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *CSL'04*, volume 3210 of *LNCS*, pages 71–84, 2004.
10. 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.
11. P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer, 1996.

12. 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.
13. 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.
14. D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.
15. 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.
16. J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–317, 1987.
17. S.-J. Lee and D. Plaisted. Eliminating duplication with the Hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
18. R. Letz and G. Stenz. Proof and model generation with disconnection tableaux. In *Proc. LPAR 2001*, volume 2250 of *LNAI*, pages 142–156, 2001.
19. W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.
20. 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.
21. J. A. N. Pérez and A. Voronkov. Encodings of bounded LTL model checking in effectively propositional logic. In *Proc. of the 21st International Conference on Automated Deduction, (CADE-21)*, volume 4603 of *Lecture Notes in Computer Science*, pages 346–361. Springer, 2007.
22. R. Pichler. Explicit versus implicit representations of subsets of the Herbrand universe. *Theor. Comput. Sci.*, 290(1):1021–1056, 2003.
23. 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.
24. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
25. S. Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.
26. 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. (to appear).
27. G. Sutcliffe. CASC-21 proceedings of the CADE-21 ATP system competition, 2007.
28. G. Sutcliffe and C. B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.