Comparing Instance Generation Methods for Automated Reasoning*

Swen Jacobs and Uwe Waldmann

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. The clause linking technique of Lee and Plaisted proves the unsatisfiability of a set of first-order clauses by generating a sufficiently large set of instances of these clauses that can be shown to be propositionally unsatisfiable. In recent years, this approach has been refined in several directions, leading to both tableau-based methods, such as the *Disconnection Tableau Calculus*, and saturation-based methods, such as *Primal Partial Instantiation* and *Resolution-based Instance Generation*. We investigate the relationship between these calculi and answer the question to what extent refutation or consistency proofs in one calculus can be simulated in another one.

1 Introduction

In recent years, there has been a renewed interest in instantiation-based theorem proving for first-order logic. Much of the recent work in this field is based on the research of Plaisted and Lee [LP92]. They showed that the interleaving of production of instances with recombination of clauses, as done in resolution, leads to duplication of work in subsequent inference steps. As a means of avoiding this duplication, they proposed the *clause linking* approach. In clause linking, *links* between complementary unifiable literals are used to generate instances of the given clauses, based on the unifier of the linked literals. Unlike in the resolution calculus, the generated instances are not recombined, but added to the set of instances as they are. As a consequence, in order to check satisfiability of the generated set of clauses, an additional SAT solving procedure is needed, which is usually in the spirit of the Davis-Putnam-Logemann-Loveland (DPLL) procedure [DLL62].

Today, there are several methods which are based to some extent on clause linking and/or DPLL. They may be distinguished by the means they use to detect unsatisfiability. There are calculi which arrange instances in a tree or a tableau, integrating an implicit satisfiability check. This approach is used by the disconnection tableau calculus [Bil96,LS01,Ste02], as well as by Baumgartner's FDPLL [Bau00] and the model evolution calculus [BT03]. Other procedures separate instance generation and satisfiability test. These usually use a monotonically growing set of clause instances and call an external SAT solver on this

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

set. Representatives of this approach are Plaisted's (ordered semantic) hyper linking [LP92,Pla94], the partial instantiation methods [HRCR02] of Hooker et al., as well as resolution-based instance generation [GK03] by Ganzinger and Korovin. As they use interchangeable SAT solvers, the main difference between these methods lies in the guidance of instance generation.

The fact that tableau-based and saturation-based instance generation methods are somehow related can be considered as folklore knowledge; it has been mentioned repeatedly in the literature (e.g., [BT03,HRCR02]). The precise relationship is, however, rather unclear. In this work, we will compare four instance generation methods that stay relatively close to the original clause linking approach, and can be seen as direct refinements of it:

The *Disconnection Calculus* (DCC) integrates the instance generation of the clause linking approach into a clausal tableau. The linking rule of DCC only allows inferences based on links which are on the current branch, strongly restricting the generation of clauses in the tableau. In this tableau structure, unsatisfiability of the given set of clauses can be decided by branch closure. Given a fair inference strategy, the calculus is refutationally complete.

Resolution-based Instance Generation (Inst-Gen) consists of a single resolution-like inference rule, representing the clause linking approach of generating instances. After a given set has been saturated under this inference rule, satisfiability of the original clause set is equivalent to propositional satisfiability and can be checked by any propositional decision procedure.

SInst-Gen is a refinement of Inst-Gen by semantic selection, based on a propositional model for the given set of clauses. Inconsistencies, which arise when extending this model to a model for the first-order clauses, are used to guide the inferences of the extension.

The *Primal Partial Instatiation* method (PPI) is treated as a special case of SInst-Gen.

After introducing the calculi, we will consider simulation of derivations from one method in the other, and we will show to what extent refutation or consistency proofs in one calculus can be simulated in the other one.

2 Introducing the Calculi

This section gives a short introduction to the methods we will compare. For a comprehensive description, we refer to Letz and Stenz [LS01,Ste02] for DCC, Ganzinger and Korovin [GK03] for SInst-Gen, and Hooker et al. [HRCR02] for PPI.

2.1 Logical Prerequisites

We use the usual symbols, notation and terminology of first-order logic with standard definitions. We consider all formulas to be in clausal normal form. This allows us to consider a formula as a set of clauses, thought to be connected by conjunction. In all of our methods, every clause of a formula is implicitly

$\forall$-quantified, while the methods themselves always work on quantifier-free formulas. Furthermore, all clauses are implicitly considered to be variable-disjoint.

If F is some formula and σ a substitution, then $F\sigma$ is an *instance* or *instantiation* of F. It is a *ground instance*, if it is variable-free. Otherwise it is a *partial instantiation*. It is a *proper instance*[1], if at least one variable is replaced by a non-variable term. A *variable renaming* on a formula F is an injective substitution mapping variables to variables. Two formulas (clauses, literals) K and L are *variants* of each other, if there is a variable renaming σ such that $L\sigma = K$. If F' is an instantiation of a formula F, then F is a *generalization* of F'. Given a set of formulas S, we say that $F \in S$ is a *most specific generalization* of F' with respect to S if F generalizes F' and there is no other formula $G \in S$ such that G generalizes F', F generalizes G, and F is not a variant of G.

For any literal L, $\overline{L}$ denotes its complement. By $\perp$ we denote both a distinguished constant and the substitution mapping all variables to this constant. We say that two literals L, L' are $\perp$*-complementary* if $L\perp = \overline{L'}\perp$. A set of clauses S is called *propositionally unsatisfiable* if and only if $S\perp$ is unsatisfiable.

2.2 The Disconnection Tableau Calculus

The *Disconnection Tableau Calculus* calculus has been developed by Billon [Bil96] and Letz and Stenz [LS01,Ste02]. In order to define development of a disconnection tableau, we need the notions of links, paths and tableaux:

A *literal occurrence* is a pair $\langle C, L\rangle$, where C is a clause and $L \in C$ a literal. If $\langle C, L\rangle$ and $\langle D, \overline{K}\rangle$ are two literal occurrences such that there is a most general unifier (mgu) σ of L and K, then the set $l = \{\langle C, L\rangle, \langle D, \overline{K}\rangle\}$ is called a *link* (between C and D). $C\sigma$ and $D\sigma$ are *linking instances* of C and D with respect to l.

A *path* P through a set of clauses (or occurrences of clauses) S is a set of literal occurrences such that P contains exactly one literal occurrence $\langle C, L\rangle$ for every $C \in S$. A path P is $\perp$*-complementary* if it contains literal occurrences $\langle C, L\rangle$ and $\langle D, \overline{K}\rangle$ such that $L\perp = K\perp$, otherwise it is *open*.

A *disconnection tableau* (tableau, for short) is a (possibly infinite) downward tree with literal labels at all nodes except the root. Given a set of clauses S, a *tableau for* S is a tableau in which, for every tableau node N, the set of literals $C = L_1, ..., L_m$ at the immediate successor nodes $N_1, ..., N_m$ of N is an instance of a clause in S. Every N_i is associated with the clause C and the literal occurrence $\langle C, L_i\rangle$.

Construction of a tableau starts from an *initial path* P_S through the set S of input clauses. The initial path may be chosen arbitrarily, but remains fixed through the construction of the tableau.

A *branch* of a tableau T is any maximal sequence $B = N_1, N_2, ...$ of nodes in T such that N_1 is an immediate successor of the root node and any N_{i+1} is an immediate successor of N_i. With every branch B we associate a path P_B containing the literal occurrences associated with the nodes in B. The union $P_S \cup P_B$ of the

[1] This definition is due to Ganzinger and Korovin [GK03] and may deviate from definitions in other areas

initial path and the path of a branch B is called a *tableau path* of B. Note that, in contrast to the initial path and the path of a branch, the tableau path may contain two literal occurrences for (two different occurrences of) the same clause.

To develop a tableau from the initial path and the empty tableau consisting of only the root node, we define the *linking rule*: Given an initial path P_S and a tableau branch B with literal occurrences $\langle C, L\rangle$ and $\langle D, \overline{K}\rangle$ in $P_S \cup P_B$, such that $l = \left\{\langle C, L\rangle, \langle D, \overline{K}\rangle\right\}$ is a link with mgu σ, the branch B is expanded with a linking instance with respect to l of one of the two clauses, say with $C\sigma$, and then, below the node labeled with $L\sigma$, the branch is expanded with the other linking instance with respect to l, $D\sigma$.

As all clauses are variable-disjoint, *variant-freeness* is required in order to restrict proof development to inferences which introduce "new" instances: A disconnection tableau T is *variant-free*, if no node N with clause C in T has an ancestor node N' with clause D in T such that C and D are variants of each other. In practice, variant-freeness is assured by two restrictions: First, a link that has already been used on the current branch may not be used again. Secondly, when a linking step is performed, variants of clauses which are already on the branch are not added to the tableau. This can result in linking steps where only one of the linking instances is added. By definition, variant-freeness does not extend to the initial path, i.e., variants of input clauses can be added to the tableau, and may be needed for completeness. This may lead to tableau paths with two literal occurrences for the same clause, in which case the occurrence from the initial path is redundant, as we have shown in Jacobs [Jac04].

Next, we define when tableau construction will terminate: A tableau branch B is *closed*, if P_B is $\perp$-complementary; if not, it is called *open*. A tableau is *closed* if it has no open branches. Similarly to the notion of variant-freeness, closure does not extend to the initial path, i.e., literals on the initial path may not be used to close a tableau branch. An exception can be made for unit clauses on the initial path, as adding those to the tableau would result in only one branch which would directly be closed. A branch B in a (possibly infinite) tableau T is called *saturated*, if B is open and there is no link on B which produces at least one linking instance which is not a variant of any clause on B. A tableau is *saturated* if either all its branches are closed or it has a saturated branch. A saturated branch represents a model for the set of input clauses of the tableau.

With these definitions, we have a sound and functional calculus. Starting from the initial path, the linking rule develops our tableau, restricted by variant-freeness. The disconnection calculus terminates if we can close all branches of the tableau, thus proving unsatisfiability of the input clauses, or if at least one branch can be saturated in finite time, thereby proving that the set of input clauses is satisfiable. If the choice of linking steps is fair, i.e., if all infinite branches are saturated, termination is guaranteed for every unsatisfiable input set.[2]

[2] For examples of open, closed and saturated tableaux, see Fig. 1, 2 and 3, respectively.

2.3 Resolution-Based Instance Generation

The *Inst-Gen* calculus is due to Ganzinger and Korovin [GK03]. It uses the following inference rule:

$$\frac{C \vee L \qquad D \vee \overline{K}}{(C \vee L)\sigma \quad (D \vee \overline{K})\sigma},$$

where σ is the mgu of K and L and a proper instantiator of either $\overline{K}$ or L.

For a set of clauses saturated under Inst-Gen, the satisfiability test can be reduced to the propositional case. As saturation may take infinitely long, however, satisfiability testing cannot be postponed until saturation is reached.

There is a formal notion of redundancy for Inst-Gen, which is, however, out of the scope of this work. The only clauses we will consider as *redundant* are variants of clauses which are already present. An inference is redundant if it only produces such variants.

SInst-Gen is an extension of Inst-Gen, which uses *semantic selection* in order to restrict the search space: Let S be a set of clauses such that $S\perp$ is satisfiable. Let $I_\perp$ be a model of $S\perp$. We define the *satisfiers* of a clause to be the set $sat_\perp(C) = \{L \in C \mid I_\perp \models L\perp\}$. Now consider *selection functions* on clauses (modulo renaming), which select for every clause in S one of its satisfiers.

Instance generation, based on a selection function sel, is defined as follows:

$$\frac{C \vee L \qquad D \vee \overline{K}}{(C \vee L)\sigma \quad (D \vee \overline{K})\sigma},$$

where σ is the mgu of K and L and both $\overline{K}$ and L are selected by sel.[3]

A selection function can be considered to represent a model for the grounded set of clauses $S\perp$. When trying to extend it to a model of S, complementary unifiable literals represent inconsistencies in the extension. Every inference of SInst-Gen resolves such an inconsistency.

In order to allow hyper-inferences, more than one satisfier can be selected. If for every selected literal in a clause, a complementary unifiable selected literal in another clause is found, then a hyper-inference produces all instances that the individual inferences between these literals would produce.

In order to ensure that unsatisfiable input sets are saturated within finite time, the choice of inferences must be *fair*. An informal idea of fairness is that any inference which is available infinitely often must either be taken or become redundant by application of other inferences at some time.

Propositional satisfiability of the generated set of clauses S' is tested after every inference step by searching for a model of $S'\perp$. If this does not exist, unsatisfiability of the input set S has been proved. If it does, another inference follows, until either no model can be found or no inferences are possible. In the latter case, the model of $S'\perp$ can be extended to a model of S without conflicts,

[3] The second condition of the Inst-Gen rule is implied by the use of semantic selection.

i.e., satisfiability of S has been shown. If the inferences are chosen in a fair manner, SInst-Gen is refutationally complete.

The primal partial instantiation (PPI) method [HRCR02][4] is equivalent to the basic case of SInst-Gen without redundancy elimination and hyper-inferences, except for the saturation strategy: PPI uses a counter which specifies the maximal term-depth of unified literals in an inference step. The counter is not reset to 0 after selection of literals is changed, and this may lead to a behaviour that is not fair with respect to the definition by Ganzinger and Korovin [Jac04]. As PPI is complete nonetheless, this may be an indication that the fairness condition of SInst-Gen is stricter than actually necessary.

3 Comparing Refutation Proofs

To compare different reasoning methods, several patterns of comparison can be used. Assuming that the objects of investigation are abstract calculi, rather than concrete implementations, one can analyze for which inputs proofs are found, how long the proofs are, or which formulas are derived during the proof. Which of these choices are appropriate for a comparison of DCC and SInst-Gen?

Both calculi are refutationally complete, and so they find (refutation) proofs for exactly the same input sets. (They may differ, however, in their ability to prove the consistency of satisfiable input sets. We will discuss this case later.)

DCC checks propositional satisfiability internally, whereas SInst-Gen uses an external program to solve this (NP-complete) subproblem. Since the number of inference steps of the external SAT solver is unknown, a meaningful comparison of the length of proofs is impossible.

The only remaining choice is to compare the internal structure of the proofs, or more precisely, the sets of clauses that are generated during the proof.

Definition 1. *A proof of method A* simulates *a given proof of method B if the instances generated by the simulating A-proof are a subset of the instances generated in the original B-proof.*

Thus, if method A can simulate all proofs of method B, B can be seen as a special case (or refinement) of A. With respect to this definition, Inst-Gen is the most general of the instance generation methods we have introduced, as it can simulate DCC, PPI and SInst-Gen proofs. That is, all of these calculi are refinements of Inst-Gen. A method that can simulate any refutation proof of another method is more general, but usually not better in the sense that it finds more proofs with limited resources (time, space). On the contrary, a strictly more general method will usually have a larger search space.

The definition of simulation is also motivated by the following two lemmas:

Lemma 2. *Let S be an unsatisfiable set of clauses, let T be a closed disconnection tableau for S. Then the set S' of all instances on the tableau is propositionally unsatisfiable.*

[4] The calculus originally presented by Hooker et al. is unsound, but can be corrected (see Jacobs [Jac04]).

Proof. Suppose S' was not propositionally unsatisfiable. Then there must be an open path through $S'\bot$. Using the literal occurrences in this path, we can identify a branch in T which selects the same literals. This branch cannot be closed, contradicting the assumption that T is closed.

The lemma implies that an SInst-Gen proof will terminate as soon as all instances from a closed disconnection tableau have been generated. Thus, we know that, if generation of these instances is possible, the requirements for our notion of simulation will be fulfilled by SInst-Gen. The next lemma does the same for the other direction of simulation:

Lemma 3. *Let S be an unsatisfiable set of clauses. If an SInst-Gen proof terminates after generating the set of instances $S' \supseteq S$, then there exists a closed tableau containing only instances from S'.*

Proof. We may simply add instances from S' to each branch of the tableau until it is closed. As there are no open paths through S', this means that every branch containing all clauses from S' must be closed.

Note, however, that neither Lemma 2 nor Lemma 3 guarantee that the required clauses can actually be generated using the construction rules of the simulating calculus. Moreover, in the disconnection calculus, even if the instances can be generated somewhere in the tableau, this does not necessarily mean that they can be generated where they are needed.

Thus, we need to compare instance generation and the guidance of proofs in both methods in order to see if simulation is possible. Instance generation itself, represented by the inference rules, is identical in both methods: the main premise is a pair of selected literals L_1, L_2 in clauses C_1, C_2 such that $L_1\sigma = \overline{L_2}\sigma$ for some mgu σ. If those are present, the instances $C_1\sigma$ and $C_2\sigma$ are generated. In both methods, one literal per clause is selected and variants of existing clauses will not be added again. Open branches correspond essentially to selection functions; if a branch is closed in DCC, then in SInst-Gen a selection function for the set of clauses is not allowed to select the same literals. The other direction, however, does not hold, because of the special role of the initial path in DCC: It is chosen and fixed at the beginning of the proof, and literals on it must not be used to close branches. There is no equivalent notion in SInst-Gen. However, adding variants of clauses from the initial path to the tableau effectively allows a different selection on and closure by input clauses, albeit only if there is a link which allows generation of the variant. The fact that literals on the initial path do not close branches may also lead to the generation of non-proper instances of input clauses. This cannot happen in SInst-Gen, as it requires two $\bot$-complementary literals to be selected. The main difference between the two approaches, however, is that instances generated in DCC will only be available on the current branch, while in SInst-Gen all instances are available at all times, i.e., regardless of the current selection.

3.1 From SInst-Gen to DCC

Theorem 4. *There exist refuting SInst-Gen proofs that cannot be simulated by any DCC proof.*[5]

Proof. The following is an SInst-Gen proof for an unsatisfiable set of clauses. We claim that it cannot be simulated by any disconnection tableau, i.e., DCC cannot finish the proof with the same set of instances as SInst-Gen, or a subset thereof. The reason for this is that the needed instances cannot be generated on all branches without generating additional instances.

The proof which we consider starts with the set of input clauses

$$\underline{\neg P(x,y)} \vee \neg P(y,z) \vee P(x,z), \quad \underline{\neg P(x,y)} \vee P(fx,c),$$
$$\underline{P(a,b)}, \quad \underline{P(b,c)}, \quad \underline{\neg P(fa,c)},$$

where a selection function is given by the underlined literals. SInst-Gen goes on to produce the following instances in the given order:

$$\neg P(a,b) \vee \neg \underline{P(b,z)} \vee P(a,z), \quad \neg P(a,b) \vee \neg P(b,c) \vee \underline{P(a,c)},$$
$$\neg P(a,c) \vee P(fa,c)$$

One can easily see that addition of the last clause makes the set propositionally unsatisfiable. SInst-Gen terminates, indicating unsatisfiability. The DCC derivation in Figure 1 tries to reproduce all steps of this SInst-Gen proof.

Within the frame we have the input clauses. The initial path of the tableau selects the same literals as the initial selection function of the SInst-Gen proof does. Links between literals are marked by dashed lines with a circle numbering the link. On the tableau, the circle above the generated clause shows the number of the link which was used. One can confirm that links number 1, 5 and 6 are equivalent to the inference steps in SInst-Gen, as they produce the same instances. Branches which are closed are marked by a $*$. We see that there is one open branch. Our definition of simulation would allow us to close this branch by generating again any of the instances from the given proof. The available links on this branch are those with numbers 2 to 4 from the initial path, as well as two additional links to $P(a,z)$ which are not displayed in the figure. However, none of those links generate one of the needed instances. Therefore, simulation has failed with this strategy.

It might still be possible that there is a tableau simulating the given SInst-Gen proof which generates instances in a different order or uses a different initial path. We have shown that even in this case simulation of the given derivation is not possible. The proof is rather lengthy and can be found in Jacobs [Jac04].

As the SInst-Gen proof given above obeys the term-depth restriction of Hooker's PPI method, it shows also that there are PPI proofs that cannot be simulated by any DCC proof:

[5] It is not known whether this result still holds if one considers some of the extensions of DCC from Stenz [Ste02].

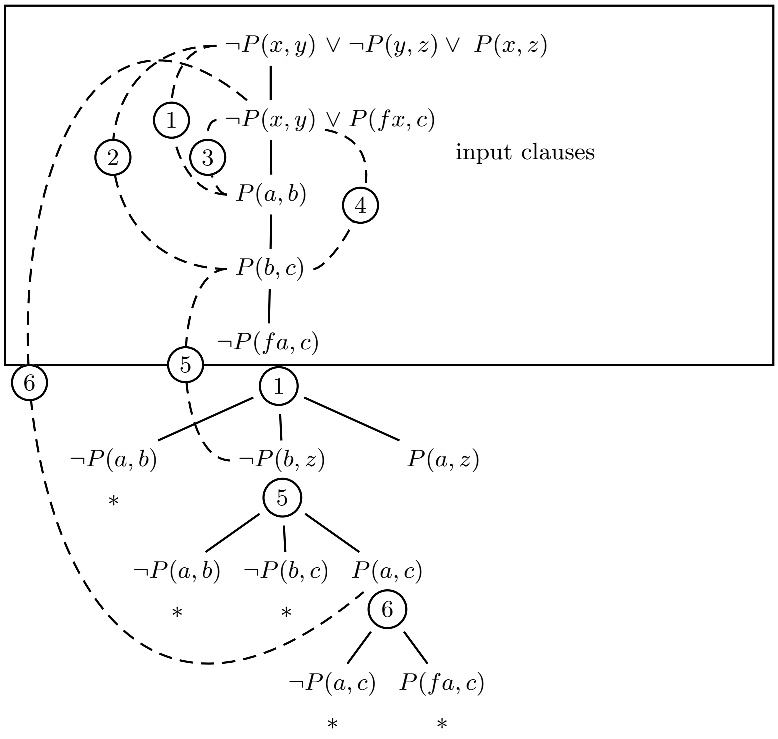

Fig. 1. DCC needs additional instances

Corollary 5. *There exist refuting PPI proofs that cannot be simulated by any DCC proof.*

3.2 From DCC to SInst-Gen

The fact that SInst-Gen proofs cannot always be simulated by DCC proofs is a consequence of the tree structure of DCC proofs. One might expect that in the other direction there is no such obstacle, but surprisingly, this is not the case.

Theorem 6. *There exist refuting DCC proofs that cannot be simulated by any SInst-Gen proof.*

Proof. Figure 2 shows a closed disconnection tableau for an unsatisfiable set of input clauses. We claim that this tableau cannot be simulated by SInst-Gen.

In order to verify this claim, let us consider all possible SInst-Gen proofs for the given set of clauses:

$$P(a,x,y,z) \vee Q(a,b,z), \quad \neg P(x,y,z,b) \vee R(x), \quad P(a,x,y,b) \vee S(y),$$
$$\neg Q(x,y,z) \vee \neg P(x,b,z,y), \quad \underline{\neg S(b)}, \quad \underline{\neg R(a)}$$

An underlined literal means that no other selection is possible in that clause. If there is no underlined literal, we consider all possible selections. There is an

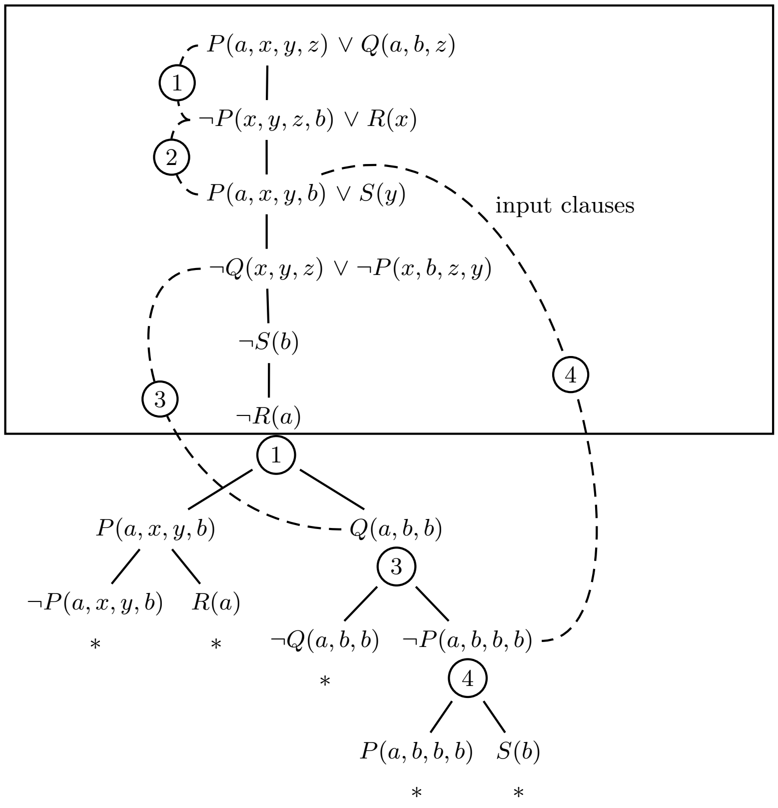

Fig. 2. SInst-Gen needs additional instances

inconsistency between $P(a,x,y,z)$ in the first and $\neg P(x,y,z,b)$ in the second clause, which is equivalent to link number 1 in the tableau (i.e., produces the same instances). Also, there are inconsistencies between $\neg P(x,y,z,b)$ in the second and $P(a,x,y,b)$ in the third clause, and between $R(x)$ in the second clause and the unit clause $\neg R(a)$, which both produce only one of these instances, $\neg P(a,x,y,b) \vee R(a)$. There are four other possible inferences on this set of clauses, but all of them produce instances which are not on the given tableau.

Let us first consider the simulating proof which only produces $\neg P(a,x,y,b) \vee R(a)$ in the first step. Except the one mentioned first above, none of the possible inferences between the input clauses is usable for a simulation. Thus, we only need to consider inconsistencies referring to the newly generated clause. Moreover, as $R(a)$ is complementary to the given unit clause $\neg R(a)$, we only need to consider inconsistencies between $\neg P(a,x,y,b)$ and literals of the given clauses. Only one new inconsistency is introduced by the new clause, related to $P(a,x,y,z)$ in the first input clause. Furthermore, both of the inferences which are admissible for a simulating proof produce the same instance, which is $P(a,x,y,b) \vee Q(a,b,b)$.

Thus, after either taking the first inference step mentioned above, or one of the other two possible inference steps and one of the two admissible subsequent inferences, we arrive at the following set of instances:

$$P(a,x,y,z) \vee Q(a,b,z), \quad \neg P(x,y,z,b) \vee R(x), \quad P(a,x,y,b) \vee \underline{S(y)},$$
$$\neg Q(x,y,z) \vee \neg P(x,b,z,y), \quad \underline{\neg S(b)}, \underline{\neg R(a)}, \underline{\neg P(a,x,y,b)} \vee R(a),$$
$$P(a,x,y,b) \vee \underline{Q(a,b,b)}$$

All admissible inferences between input clauses have been carried out and selection is fixed on all of the generated instances. Thus, we have only one possible inference, which is between $Q(a,b,b)$ in the last and $\neg Q(x,y,z)$ in the fourth clause. This step is equivalent to linking step number 3 in the tableau, generating the instance $\neg Q(a,b,b) \vee \neg P(a,b,b,b)$.

Now, we have to select $\neg P(a,b,b,b)$ in the last instance, which only gives us one new inference, connected to $P(a,x,y,z)$ in the first clause. Note that the inference equivalent to linking step number 4 is not possible, as $P(a,x,y,b)$ is $\perp$-complementary to $\neg P(a,x,y,b)$, which has to be selected in the seventh clause. The new inference generates $P(a,b,b,b) \vee Q(a,b,b)$, which is not on the given tableau. At this point, there is no inference which does not violate our simulation property, which means that simulation has failed.

This result also holds if hyper-inferences are allowed in SInst-Gen, as one can easily verify that the possible hyper-inferences either produce instances which violate simulation, or only produce the same instances as the standard inferences.

3.3 Weak Simulation

We have shown that simulation of refutational SInst-Gen (or PPI) proofs by DCC, or vice versa, fails in general. We can get positive simulation results, however, if the definitions of the calculi are slightly changed and if the definition of simulation is modified in the following way:

Definition 7. *A proof by method A* simulates *a given proof by method B* weakly*, if every instance generated by the simulating A-proof is a generalization of an instance generated in the original B-proof.*

Recently, Letz and Stenz [LS04] introduced a more general version of the linking rule, which uses instead of standard unification a special form of a unifier: Given two literals L and L', a substitution σ is called a $\perp$-unifier of L and L' if $L\sigma\perp = L'\sigma\perp$. σ is called a most general $\perp$-unifier of L and L' if it is more general than every $\perp$-unifier of L and L'. E.g., $\perp$-unification of $P(a,x,y)$ and $P(x',y',y')$ results in $P(a,x,y)$ and $P(a,y',y')$.

It has already been stated [GK03] that the degree of instantiation of an SInst-Gen inference can be chosen flexibly, as long as at least one variable is instantiated properly. As for properly instantiated variables there is no difference between standard and $\perp$-unification, we can safely assume that we can use $\perp$-unification also for SInst-Gen.

Theorem 8. *Let S be an unsatisfiable set of clauses, S' a set of instances of clauses from S such that $(S \cup S')\bot$ is unsatisfiable. If M is a finite subset of $S \cup S'$ such that $M\bot$ is unsatisfiable, then SInst-Gen with $\bot$-unification can prove unsatisfiability of S by only generating generalizations of clauses from M.*

Proof. As the set of all generalizations of clauses in M is finite (up to renaming), it is not necessary to consider infinite derivations using such clauses. So the only way how the construction of an SInst-Gen proof using generalizations of clauses in M can fail is that at some point of the proof, SInst-Gen has generated a set of clauses M_1 such that $M_1\bot$ is satisfiable and every possible inference on M_1 results in generation of an instance which is not a generalization of any clause in M. As $M_1\bot$ is satisfiable, we can choose a selection function sel on M_1. Every clause $C\sigma \in M$ has at least one most specific generalization C with respect to M_1. Suppose we choose one of these most specific generalizations for every $C\sigma \in M$ and select the literal $L\sigma \in C\sigma$ if L is selected by sel in C. As $M\bot$ is unsatisfiable, we must have selected at least one pair of $\bot$-complementary literals, say $L_1\sigma_1 \in C_1\sigma_1$ and $L_2\sigma_2 \in C_2\sigma_2$.

Thus, in M_1, sel selects $L_1 \in C_1$ and $L_2 \in C_2$. As $L_1\sigma_1$ and $L_2\sigma_2$ are $\bot$-complementary, we can state that L_1 and L_2 are complementary unifiable, say by τ. As they are not $\bot$-complementary, there is an SInst-Gen inference with $\bot$-unification between them. The substitution used in this inference is a most general $\bot$-unifier, therefore the clauses produced by this inference will also be generalizations of clauses from M. Note that this would not hold for SInst-Gen without $\bot$-unification. The inference generates at least one proper instance with respect to the premises, say $C_1\tau$ is a proper instance of C_1. There cannot be a variant of $C_1\tau$ in M_1, as C_1 was chosen to be a most specific generalization of $C_1\sigma$. Thus, we have produced a new generalization of a clause from M, contradicting our assumption that no such inference is possible.

Corollary 9. *For every refuting DCC proof (with or without $\bot$-unification) there exists a weakly simulating SInst-Gen proof (with $\bot$-unification).*

Proof. For a DCC proof that shows the unsatisfiability of a set of clauses S, let S' be the finite set of all clauses on the DCC tableau. Since $(S \cup S')\bot$ is unsatisfiable, we can apply the previous theorem.

Theorem 10. *Let S be an unsatisfiable set of clauses, S' a set of instances of clauses from S such that $(S \cup S')\bot$ is unsatisfiable. If M is a finite subset of $S \cup S'$ such that $M\bot$ is unsatisfiable, then DCC with $\bot$-unification can prove unsatisfiability of S by only generating generalizations of clauses from M.*

Proof. Again, the fact that the set of all generalizations of clauses in M is finite ensures that unfair derivations need not be considered. Suppose a DCC tableau for S has an open branch B that cannot be extended without generating instances that are not generalizations of clauses in M. Every clause in $C\sigma \in M$ has at least one most specific generalization C with respect to the clauses on the tableau path $P_S \cup P_B$. We select in each $C\sigma \in M$ the literal corresponding to the literal of C on the tableau path. As $M\bot$ is unsatisfiable, $\bot$-complementary literals $L\sigma \in C\sigma$ and $K\tau \in D\tau$ are selected. Thus the literals L and K of the most specific generalizations C and D are on the tableau path. The DCC inference with $\bot$-unification from C and D uses a most general $\bot$-unifier, so the instances produced by this inference are again generalizations of $C\sigma$ and $D\tau$; moreover, at least one of them is a proper instance of a premise. This instance cannot be a variant of a clause on the tableau path, since C and D were chosen as most specific generalizations of $C\sigma$ and $D\tau$. Therefore, the tableau can be extended with a generalization of a clause in M, contradicting our assumption.

Corollary 11. *For every refuting SInst-Gen or PPI proof (with or without $\bot$-unification) there exists a weakly simulating DCC proof (with $\bot$-unification).*

4 Comparing Consistency Proofs

4.1 From SInst-Gen to DCC

The case of consistency proofs differs in several aspects from the case of refuting proofs. First, it is clear that no theorem proving method is guaranteed to terminate for satisfiable sets of clauses. Second, in order to declare a set of clauses unsatisfiable, SInst-Gen must check *all* propositional interpretations for the given set of clauses and show that none of them is a model. In contrast to this, termination on a satisfiable set of clauses only depends on *one* interpretation which can be extended to a first-order model of the input clauses. Essentially, the same holds for DCC. Simulation can therefore be based on the final set of clauses and selection function. In the following, we will show that this fact enables us to simulate any SInst-Gen proof which terminates on satisfiable input by DCC.

Theorem 12. *Let S be a satisfiable set of input clauses, $S \cup S'$ a finite set of clauses saturated under SInst-Gen proof with selection function* sel*. Then the given consistency proof can be simulated by DCC.*

Proof. We prove our claim by induction on the number of proof steps of the simulating proof, where a proof step consists of both the selection of literals and the generation of instances. Simulation is not based on the steps of the given proof, but only on the final set of clauses $S \cup S'$ and the selection function sel. We will show that every step of the simulating proof produces only instances from $S \cup S'$, while the tableau path we follow is always equivalent to sel for the clauses which are on this path.

First, we choose the initial path P_S of our simulating proof to select the same literals as sel on S. Then, every link on P_S can only produce instances from $S \cup S'$, as otherwise there would be an inconsistency of sel. Thus, we may carry out an arbitrary link from those which are available on P_S, resulting in a set of clauses which is a subset of $S \cup S'$.

Now, suppose an arbitrary number of steps has been carried out by our simulating proof, always following a tableau path which selects the same literals as sel for clauses on the path and only generating instances from $S \cup S'$. We can extend the path from the last step such that it selects the same literals as sel on the new instances. As sel does not select $\bot$-complementary literals, the path must also be open. By the same argument as above, we may again carry out any possible linking step, producing only instances from $S \cup S'$.

In this way, we carry out all possible linking steps. As we choose our tableau path such that only instances from $S \cup S'$ are produced, the process must terminate after a finite number of steps. As termination of the process means that we have saturated the current path, we can state that we have reached our goal to simulate the given proof.

It may happen that a linking step adds two instances, but sel is such that we cannot select a tableau path which is equivalent to the selection function *and* considers both of these clauses. This happens if both of the selected literals are not linked literals of the linking step that produces them. In this case, our tableau path can only consider one of the produced clauses. However, as we want to saturate the branch we are following, missing clauses and links are not a problem, but a benefit in this case.

4.2 From DCC to SInst-Gen

We have shown that every SInst-Gen consistency proof can be simulated by DCC. The reverse, however, does not hold: Figure 3 shows a saturated disconnection tableau for a satisfiable set of input clauses. Saturation of the tableau is achieved by generating a single non-proper instance of an input clause.

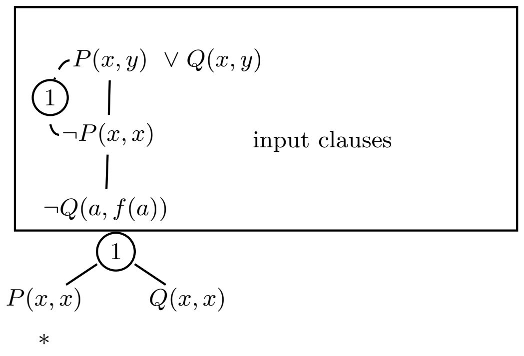

Fig. 3. Saturated disconnection tableau

An SInst-Gen proof for the given set of clauses would have to select $Q(x, y)$ in the first input clause, as $\bot$-complementary literals must not be selected. Thus, we have an inconsistency between this literal and $\neg Q(a, f(a))$, which produces an instance we do not have on the tableau. Without producing this instance however, satisfiability of the input set cannot be detected by SInst-Gen.

If $\perp$-unification is used for both DCC and SInst-Gen, examples like this one are obsolete, as only proper instances or variants of input clauses will be added to the tableau. However, even in this case simulation is in general not possible, as we have demonstrated by a more complicated counterexample [Jac04].

As for refutation proofs, this result still holds when hyper-inferences are allowed in the simulating SInst-Gen proof.

5 Conclusions

We have compared the four instance generation methods DCC, Inst-Gen, SInst-Gen, and PPI. Inst-Gen, which does not make any attempt to restrict the search space, is obviously the most general of these calculi: any DCC, SInst-Gen, or PPI proof can be simulated by an Inst-Gen proof.

PPI is essentially a special case of SInst-Gen, except for its term-depth-based saturation strategy, which ensures completeness of the calculus but is not subsumed by the fairness criterion of SInst-Gen. It would be interesting to search for a more relaxed notion of fairness which allows all possible strategies of both PPI and SInst-Gen.

For DCC and SInst-Gen, we have demonstrated that, for refutation proofs, simulation between (the basic versions of) the methods is in general not possible. This implies in particular that neither of the methods can be considered as a special case of the other. In case of consistency proofs, we have shown that SInst-Gen proofs that terminate on satisfiable input can always be simulated by DCC, while this does not hold for the other direction. All of these results still hold when SInst-Gen is allowed to use hyper-inferences.

We face a very different situation when we consider weak simulation, so that we can not only use clauses from the given proof but also generalizations thereof. We have shown that DCC and SInst-Gen with $\perp$-unification can weakly simulate each other; in fact we conjecture that DCC and SInst-Gen with $\perp$-unification can weakly simulate any instance-based calculus.

For DCC and SInst-Gen there are various refinements which are out of the scope of this work. It is not clear how our results would translate to the refined calculi, e.g. DCC with lemma generation and subsumption or SInst-Gen with redundancy elimination. Additionally, for both methods extensions to equality reasoning are available. A comparison of the different approaches to these refinements and extensions might give more useful insights on the relation between DCC and SInst-Gen.

References

[Bau00] Peter Baumgartner. FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In *CADE-17, LNAI 1831*, pages 200–219. Springer, 2000.

[Bil96] Jean-Paul Billon. The disconnection method: a confluent integration of unification in the analytic framework. In *Tableaux 1996, LNAI 1071*, pages 110–126. Springer, 1996.

[BT03] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In *CADE-19, LNAI 2741*, pages 350–364. Springer, 2003.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications ACM*, 5:201–215, 1962.

[GK03] Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *LICS-03*, pages 55–64, Ottawa, Canada, 2003. IEEE.

[HRCR02] John N. Hooker, Gabriela Rago, Vijay Chandru, and Anjul Rivastava. Partial instantiation methods for inference in first-order logic. *Journal of Automated Reasoning 28*, pages 371–396, 2002.

[Jac04] Swen Jacobs. Instance generation methods for automated reasoning. Diploma Thesis, Universität des Saarlandes, 2004. Available at http://www.mpi-sb.mpg.de/~sjacobs/publications.

[LP92] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

[LS01] Reinhold Letz and Gernot Stenz. Proof and model generation with disconnection tableaux. In *LPAR 2001, LNAI 2250*, pages 142–156. Springer, 2001.

[LS04] Reinhold Letz and Gernot Stenz. Generalised handling of variables in disconnection tableaux. In *IJCAR 2004, LNCS 3097*, pages 289–306. Springer, 2004.

[Pla94] David A. Plaisted. Ordered semantic hyper-linking. Technical Report MPI-I-94-235, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994.

[Ste02] Gernot Stenz. *The Disconnection Calculus*. PhD thesis, TU München, 2002.