INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians-
Universität
München

Comparison of Two Complementary Herbrand Model Generators

Heribert Schütz

http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-18, Dezember 1996

Comparison of Two Complementary Herbrand Model Generators

Heribert Schütz

Universität München, Institut für Informatik
Oettingenstr. 67, D-80538 München, Germany
Heribert.Schuetz@informatik.uni-muenchen.de

Abstract. This paper investigates and compares two approaches for generating Herbrand models of clausal first-order theories: Satchmo by R. Manthey and F. Bry and the method introduced by C. Fermüller and A. Leitsch. A uniform description of the two approaches is given for this purpose.
The two approaches turn out to be able to handle similar classes of theories in principle. However, theories are typically handled efficiently by one of the approaches and inefficiently by the other. In this sense, the approaches are complementary. A characterization of the classes of theories for which each of the approaches is better is given and guidelines for a combination of the approaches are derived from this characterization.

1 Introduction

In the last years in the automated deduction community there has been increasing interest in *model generation*, i.e., in methods for automatic construction of interpretations that satisfy a given theory. This research direction complements the traditional direction of *theorem proving*. Model generation is useful in the area of automated theorem proving: Automatically constructed models can be used for finding conjectures or as counterexamples for theorem-prover input with bugs. They can even be used to guide a theorem prover [15]. Model generation is also useful in the areas of disjunctive logic programming and deductive databases for query answering [9, 13] and as an implementation method for abductive reasoning [7].

For first-order logic the satisfiability of a theory, i.e., the existence of a model, is not semi-decidable. Therefore a model generation procedure cannot be expected to find a model for an arbitrary first-order theory. Consequently, model generators typically impose some restrictions either on the class of theories they can handle or on the class of models they can construct.

On the one hand, there are model generators that search for models with a given finite domain size (e.g., "Finder" [14]). The techniques used by most of these systems differ considerably from the techniques used in first-order theorem proving, and are more related to techniques for propositional reasoning.

On the other hand, there are model generators that search for Herbrand models, which is not a severe restriction on the class of models. Therefore these systems cannot handle arbitrary first-order theories. Herbrand model generators typically use variants of techniques that have been developed for theorem provers. The approaches to model generation by R. Manthey and F. Bry [11] ("Satchmo"), R. Caferra and N. Zabel [4], T. Tammet [17], and C. Fermüller and A. Leitsch [5] are all based on some variant of resolution.

In particular, Manthey and Bry as well as Fermüller and Leitsch use positive hyperresolution [12]. The former approach uses very restricted versions of positive hyperresolution and subsumption and also applies branch splitting as in semantic tableaux [16, 6]. The latter uses full hyperresolution and subsumption and a non-standard selection rule and avoids branch splitting.

This paper investigates and compares these two approaches. In order to facilitate this comparison and to make the differences and similarities more visible, the approaches are first described in a uniform way. In the comparison it turns out that the classes of applications that can in principle be handled with each of the two approaches are very similar. But it also turns out that the two approaches are complementary in the sense that applications that are handled efficiently by one of the approaches are typically not handled efficiently by the other approach and vice versa. This suggests to combine the approaches to achieve a method that is able to handle a larger class of applications efficiently. Guidelines for such a combination will be derived from a characterization of the classes of theories that are handled more efficiently by either of the approaches.

2 Two Approaches for Model Generation

This section contains descriptions and analyses of the two calculi on which the work presented later on will be based. The descriptions given here differ from the original ones (in particular in Section 2.2) in order to make similarities and differences more visible by a more uniform representation.

Both approaches require their input theory to be given in clausal form. Input clauses will be written in implication form, i.e., as $B_1 \wedge \ldots \wedge B_n \rightarrow H_1 \vee \ldots \vee H_m$ with atoms B_i and H_j. Variables are implicitly universally quantified over the entire clause. $B_1 \wedge \ldots \wedge B_n$ and $H_1 \vee \ldots \vee H_m$ are called the *body* and the *head* of the clause. An empty body is omitted together with the implication symbol. An empty clause or an empty head is written as "$\perp$". A clause is called *positive* if its body is empty and *negative* if its head is empty. A term, an atom, or a clause without variables is said to be *ground*.

Substitutions are considered as functions and written as prefix rather than postfix operators.

2.1 PUHR Tableaux

The *positive unit hyperresolution (PUHR) tableau* calculus has been introduced by F. Bry and A. Yahya [2, 3] as a formalization of Satchmo, the theorem prover and model generator introduced by R. Manthey and F. Bry [11]. It applies to

$$\frac{\begin{array}{c} A_1 \\ \vdots \\ A_n \\ i : B_1 \wedge \ldots \wedge B_n \rightarrow H \end{array}}{\sigma(H)} \qquad\qquad \frac{A_1 \vee \ldots \vee A_n}{A_1 \mid \ldots \mid A_n}$$

if $A_1 = \sigma(B_1)$,...,$A_n = \sigma(B_n)$ — if none of $A_1, \ldots, A_n$ occurs in the branch;

(PUHR) — (Splitting)

Fig. 1. Rules of the PUHR-tableau calculus

sets of *range-restricted* clauses, i.e., clauses in which every variable appearing in the head appears also in the body.

PUHR tableaux are trees in which every node except for the root is labeled with a positive ground clause. PUHR tableaux for a given clause set are built starting from the tree consisting of a single unlabeled node, by repeated application of the following two expansion rules:

(PUHR) Let some branch contain (among others) nodes with atomic (i.e., positive unit) labels $A_1, \ldots, A_n$ (the *electrons*) and let there be a clause $B_1 \wedge \ldots \wedge B_n \rightarrow H$ (the *nucleus* with atoms B_i and a positive clause H) in the input theory such that for some substitution σ it holds that $A_1 = \sigma(B_1)$, $\ldots$, $A_n = \sigma(B_n)$. Then a node labeled with $\sigma(H)$ (the *(hyper-)resolvent*) can be appended as the single child to the final node of this branch.

(Splitting) Let some branch contain a node labelled with a positive clause $A_1 \vee \ldots \vee A_n$ (where the A_i are atoms and $n > 1$) but no nodes labeled with any of the atoms $A_1, \ldots, A_n$. Then n nodes labeled with $A_1, \ldots, A_n$ can be appended as children to the final node of this branch.

The expansion rules are visualized in Fig. 1. In the PUHR rule the clause from the input theory, which is not taken from the branch, is tagged by "i :". The splitting rule is a variant of the usual β-rule for tableau calculi with a regularity condition.

The notion of PUHR tableaux is extended to infinite trees (with branches of length $\leq \omega$) in the usual way [16, 6]. Notice that the range-restriction of the input clauses ensures that all clauses labelling nodes in a PUHR tableau will in fact be ground.

A branch containing a node labeled with the empty clause is said to be *closed*. A PUHR tableau is *closed* if all its branches are closed. A branch or a PUHR tableau is *open* if it is not closed. An open branch is *saturated* if for every ground instance of a clause in the input theory some body atom does not occur as a label of a node in the branch or its head does, and if furthermore for every positive clause $A_1 \vee \ldots \vee A_n$ occurring as a node label on the branch one of the atoms

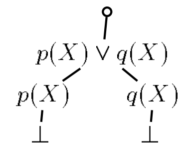

Fig. 2. A closed PUHR tableau for the non–range-restricted satisfiable clause set $\{p(X) \vee q(X), p(a) \rightarrow \bot, q(b) \rightarrow \bot\}$.

$A_1, \ldots, A_n$ occurs, too. A PUHR tableau is *saturated* if all of its open branches are saturated.

PUHR tableaux for a set $\mathcal{C}$ of input clauses have the following properties [2]:

1. The set of atomic labels in a saturated open branch of a PUHR tableau is a Herbrand model[1] of $\mathcal{C}$ ("model soundness"). As a consequence, if $\mathcal{C}$ is unsatisfiable, then every saturated PUHR tableau for $\mathcal{C}$ is closed ("refutation completeness").
2. Every model of $\mathcal{C}$ satisfies all node labels of some open branch of every PUHR tableau for $\mathcal{C}$. As a consequence, if $\mathcal{C}$ is satisfiable, then there is no closed PUHR tableau for $\mathcal{C}$ ("refutation soundness").
3. Every minimal Herbrand model of $\mathcal{C}$ occurs in every saturated PUHR tableau for $\mathcal{C}$ as the set of the atomic node labels of some open branch ("minimal model completeness").

Property 1 is an immediate consequence of the definition of a *saturated* branch. Property 2 is proved by induction over a sequence of expansion steps. The induction step for the PUHR rule is a trivial consequence of the well-known soundness of hyperresolution. In the induction step for the splitting rule the range-restriction condition is used. Splitting of a non-ground clause might not be correct. Consider the input clause set $\{p(X) \vee q(X), p(a) \rightarrow \bot, q(b) \rightarrow \bot\}$, which is not range-restricted and satisfied by the Herbrand model $\{p(b), q(a)\}$ with the domain $\{a, b\}$. A closed PUHR tableau for $\mathcal{C}$ is given in Fig. 2. The problem is in the splitting step, which essentially implements a transition from the formula $\forall X(p(X) \vee q(X))$ to the formula $\forall X p(X) \vee \forall X q(X)$,[2] which is not a consequence of the original formula.

The details of property 3 are not investigated here. Nevertheless it should be mentioned that with a modification of the PUHR-tableau calculus [2] it is possible to efficiently generate tableaux for which there is a one-to-one correspondence between the open branches and the set of minimal models of the input clauses.

Because of the properties of PUHR tableaux given above it is desirable to construct saturated tableaux. Every PUHR tableau can in fact be expanded to

[1] As usual in the area of logic programming, a Herbrand interpretation is identified with the set of ground atoms it satisfies.

[2] Note that there is no connection beween variables in different node labels as in *free variable tableaux* [6].

a saturated PUHR tableau, albeit perhaps only after ω steps. If the set of input clauses is unsatisfiable, then every PUHR tableau can be expanded to a closed PUHR tableau in a finite number of expansion steps. Some fairness criterion as known from standard tableau methods for the selection of expansion steps ensures that a saturated or even a closed PUHR tableau is reached by a sequence of expansion steps. Therefore it is not necessary to perform backtracking in the traversal of the search space (which is the set of all PUHR tableaux for the set of input clauses).

2.2 Model Generation Based on Subsumption, Hyperresolution, and Selection

The model generation method by C. Fermüller and A. Leitsch [5] is based on positive hyperresolution (PHR), subsumption, and a non-standard selection rule. This method will be referred to as the *FL method.* Instead of trees, it manipulates sets of positive clauses, which will be called call *pc-sets.* The clauses in pc-sets are assumed to be in some normal form w.r.t. associativity, commutativity, and idempotency of the disjunction. This allows to remove some clauses as duplicates. Pc-sets are built starting from the empty set by repeated application of the following three transformation rules:

(Subsumption) Let the current pc-set contain two positive clauses C and C' such that $\sigma(C)$ is a subclause of C' for some substitution σ. C is said to *subsume* C'. If C and C' are different clauses, then C' can be removed from the pc-set.

(PHR) Let the current pc-set contain (among others) positive clauses $L_1 \vee R_1, \ldots, L_n \vee R_n$ (the *electrons* with non-empty subclauses $L_1, \ldots, L_n$ and possibly empty rests $R_1, \ldots, R_n$) and let there be an input clause $B_1 \wedge \ldots \wedge B_n \rightarrow H$ (the *nucleus* with atoms B_i and a positive clause H). Furthermore, let there be a tuple $(\sigma, \sigma_1, \ldots, \sigma_n)$ of substitutions such that $\sigma_1(L_1) = \sigma(B_1)$ and ... and $\sigma_n(L_n) = \sigma(B_n)$ and let $(\sigma, \sigma_1, \ldots, \sigma_n)$ be a most general[3] tuple of substitutions with this property. Then a condensation[4] of $\sigma(H) \vee \sigma_1(R_1) \vee \ldots \vee \sigma_n(R_n)$ (the *(hyper-)resolvent*) can be added to the pc-set. $(\sigma, \sigma_1, \ldots, \sigma_n)$ is the *unifier* of this PHR step.
Note that—as usual in (hyper-)resolution—commutativity and associativity of the disjunction may be used for finding electrons, i.e., the electrons may in fact be permutations of $L_i \vee R_i$. In the conditions $\sigma_i(L_i) = \sigma(B_i)$ the idempotency of the disjunction may be used (and must be used for a non-unit L_i). This use of idempotency implements implicit factoring on the electrons.

(Selection) Let the current pc-set have the property that none of its members is subsumed by another member, and that every possible application of the

[3] An n-tuple $(\sigma_1, \ldots, \sigma_n)$ of substitutions is *more general* than another n-tuple $(\tau_1, \ldots, \tau_n)$ of substitutions iff there is a substitution ρ with $\tau_1 = \rho \circ \sigma_1$ and ... and $\tau_n = \rho \circ \sigma_n$.

[4] A *condensation* of a clause is a smallest subclause that is subsumed by the full clause [8].

$$
\frac{\begin{array}{c} L_1 \lor R_1 \\ \vdots \\ L_n \lor R_n \\ i: B_1 \land \ldots \land B_n \rightarrow H \end{array}}{\sigma(H) \lor \sigma_1(R_1) \lor \ldots \lor \sigma_n(R_n)}
$$

$$
\frac{\begin{array}{c} C \\ \sigma(C) \lor D \end{array}}{\cancel{\sigma(C) \lor D}}
$$

(Subsumption)

if $(\sigma, \sigma_1, \ldots, \sigma_n)$ is a most general tuple of substitutions such that $\sigma_1(L_1) = \sigma(B_1), \ldots, \sigma_n(L_n) = \sigma(B_n)$

(PHR)

$$
\frac{A_1 \lor \ldots \lor A_n}{A_i}
$$

if the pc-set is stable;

(Selection)

Fig. 3. Rules of the FL method

PHR rule would generate a hyperresolvent that is subsumed by some member of the pc-set. Such a pc-set is called *stable*. Furthermore, let the current pc-set contain some positive clause $A_1 \lor \ldots \lor A_n$ (with atoms A_i and $n > 1$). Then one of the atoms A_i may be added to the pc-set.[5]

The FL method terminates if the pc-set is stable and all members of the pc-set are units. The transformation rules are visualized in Fig. 3. Again, the tag "i :" distinguishes input from derived clauses. Removal of a clause is denoted by striking it out.

A clause is said to be *decomposed* if there is no variable that occurs in more than one literal of the clause. Then a class *PDC* of input clause sets is defined as follows: A clause set $\mathcal{C}$ belongs to PDC iff the set of all positive clauses that can be constructed from clauses in $\mathcal{C}$ by repeated application of the PHR rule (without subsumption or selection) is finite and every clause in this set is decomposed.

Membership in PDC is undecidable. There are, however, certain subclasses of PDC [5], for which membership is decidable, e.g.:

- A clause set $\mathcal{C}$ belongs to class *PVD+* iff every clause C in $\mathcal{C}$ has the following properties:
  - C is range-restricted (see Section 2.1).
  - For every variable that occurs in the head and in the body of C the maximal nesting level in the head is not higher than the maximal nesting level in the body.

    The *nesting level* of a variable occurrence is the number of terms of which this variable occurrence is a part.
- A clause set $\mathcal{C}$ belongs to class *OCC1N+* iff every clause C in $\mathcal{C}$ has the following properties:
  - No variable occurs more than once in the head of C.

[5] $A_1 \lor \ldots \lor A_n$ can then be removed by a subsumption step.

- For every variable that occurs in the head and in the body of C the maximal nesting level in the head is not higher than the minimal (!) nesting level in the body.

The FL method can be applied to clause sets in PDC. More precisely, the following properties hold for an input clause set $\mathcal{C}$:

1. If $\mathcal{C}$ is unsatisfiable, then the empty clause can be inserted into the pc-set without an application of the selection rule.
2. If $\mathcal{C}$ belongs to PDC and is satisfiable, then every generated pc-set is satisfiable.
3. The set of ground instances of the atoms in any terminal pc-set is a Herbrand model of $\mathcal{C}$.
4. If the input clause set $\mathcal{C}$ belongs to PDC and is satisfiable, then any sequence of transformation steps which does not perform the same PHR step infinitely often, leads to a terminal pc-set after a finite number of steps.

Property 1 is the well-known completeness result for positive hyperresolution with subsumption. Property 3 is an immediate consequence of the stability of terminal pc-sets. Properties 2 and 4 have been proven in the paper introducing this method [5] and adapted to the presentation of the method in the current paper.

Let us now have a closer look at the question how and why the FL method works for clause sets in PDC and in particular at property 2. Let $\mathcal{C}$ be an input clause set in PDC, let $\mathcal{P}$ be a pc-set for $\mathcal{C}$, and let $\mathcal{P}'$ be a pc-set that is generated from $\mathcal{P}$ by one transformation step τ. If τ is a subsumption step, then $\mathcal{P}'$ is trivially a logical consequence of $\mathcal{C} \cup \mathcal{P}$. If τ is a PHR step, then $\mathcal{P}'$ is also a logical consequence of $\mathcal{C} \cup \mathcal{P}$ because of the soundness of hyperresolution. If τ is a selection step, then $\mathcal{P}'$ is not necessarily a logical consequence of $\mathcal{C} \cup \mathcal{P}$. For example, consider the situation $\mathcal{P} = \mathcal{C} = \{a \vee b\}$, from which a selection step can generate $\mathcal{P}' = \{a\}$.

Nevertheless a selection step preserves satisfiability, i.e., $\mathcal{C} \cup \mathcal{P}'$ has a model if $\mathcal{C} \cup \mathcal{P}$ has one. For this property it is essential that the selection rule may only be applied to a pc-set $\mathcal{P}$ which would not be changed by a subsumption step or a PHR step with subsequent subsumption. For example, consider the situation $\mathcal{C} = \{a \vee b, a \to \bot\}$ and $\mathcal{P} = \{a \vee b\}$, from which a selection step without the stability condition could generate $\mathcal{P}' = \{a\}$. Obviously $\mathcal{C} \cup \mathcal{P}$ is satisfiable whereas $\mathcal{C} \cup \mathcal{P}'$ is not. However, in situations where the stability condition holds it can be argued as follows:

For the propositional case, assume that a selection step generating an atom A from a positive clause $A \vee R$ would not preserve satisfiability. Then, by the completeness of positive hyperresolution, the empty clause could be derived from $\mathcal{C} \cup \mathcal{P}'$ by PHR steps. Notice that for every clause C that can be derived from $\mathcal{C} \cup \mathcal{P}'$ by PHR steps the clause C itself or the clause $C \vee R$ can be derived from $\mathcal{C} \cup \mathcal{P}$ by PHR steps. Because of the stability of $\mathcal{P}$, some member of $\mathcal{P}$ subsumes C or $C \vee R$, i.e., it subsumes $C \vee R$. Because the empty clause can be derived from $\mathcal{C} \cup \mathcal{P}'$, R is subsumed by a member of $\mathcal{P}$. This clause also subsumes $A \vee R$

and is different from $A \vee R$, and so $A \vee R$ does not occur in $\mathcal{P}$, again due to the stability of $\mathcal{P}$. This is a contradiction to our assumption.

For the first-order case the argumentation becomes slightly more complex. It relies on the use of condensation in PHR steps, which avoids the accumulation of several copies of R with new variables in the clauses derived from $\mathcal{C} \cup \mathcal{P}$. Furthermore it is required that $A \vee R$ is decomposed, which is true because $\mathcal{C}$ is in PDC. Consider the situation $\mathcal{C} = \{p(X) \vee q(X), p(a) \rightarrow \bot, q(b) \rightarrow \bot\}$, $\mathcal{P} = \{p(X) \vee q(X), q(a), p(b)\}$, where splitting might lead to $\mathcal{P}' = \{p(X), q(a), p(b)\}$. Here $\mathcal{C} \cup \mathcal{P}'$ is unsatisfiable whereas $\mathcal{C} \cup \mathcal{P}$ has the Herbrand model $\{p(b), q(a)\}$ with domain $\{a, b\}$.

These considerations should give an intuition why the selection rule preserves satisfiability. For the formal proof see [5].

Finally, it should be mentioned that Fermüller and Leitsch have also presented a procedure for generating models with a finite domain from terminal pc-sets.

3 Comparison of the Two Approaches

In this section the two approaches for model generation are compared with a focus on two questions: Which types of problems can in principle be solved by the two approaches? Which approach is preferable for which type of application?

3.1 Range of Applicability

According to the descriptions in the previous section, PUHR tableaux can deal with all range-restricted clause sets, whereas the FL method can also handle certain non–range-restricted clause sets but requires that only a finite number of clauses can be derived by positive hyperresolution. Actually, the classes of clauses that can be handled by the two approaches are far more similar.

Clauses that do not fulfill some nesting condition (as it has been given for the classes PVD+ and OCC1N+), are dangerous for PUHR tableaux, too. Consider the input clause set $\mathcal{C} = \{p(a), p(X) \rightarrow p(f(X))\}$. The only minimal Herbrand model of $\mathcal{C}$ is $\{p(f^i(a)) | i \in \mathbb{N}\}$. Since it is infinite, it cannot be computed within a finite number of steps by the PUHR tableau calculus. For practical applications only finite PUHR tableaux can be used.

Now consider a clause set $\mathcal{C}$ that is not range-restricted. The PUHR tableau calculus can be applied to $\mathcal{C}$, although we do not (yet) know how to assign a meaning to the generated tableaux. Of course, now the electrons in PUHR steps may contain variables. Therefore the one-way matching in PUHR steps must be replaced by full unification. The PUHR step is modified as follows:

(PUHR+) Let some branch contain nodes with atomic labels $A_1, \ldots, A_n$ (the *electrons*) and let there be an input clause $B_1 \wedge \ldots \wedge B_n \rightarrow H$ (the *nucleus* with atoms B_i and a positive clause H). Furthermore, let $(\sigma, \sigma_1, \ldots, \sigma_n)$ be a most general tuple of substitutions such that $\sigma_1(A_1) = \sigma(B_1)$ and … and

$\sigma_n(A_n) = \sigma(B_n)$. Then a node labeled with $\sigma(H)$ (the *(hyper-)resolvent*) can be appended as the single child to the final node of this branch.

Because of the presence of variables in node labels the regularity condition in the splitting step should also be changed to become a "unit subsumption" test:

(Splitting+) Let some branch contain a node with label $A_1 \vee \ldots \vee A_n$ (with atoms A_i and $n > 1$) but no node labeled with *an atom subsuming* any of the atoms $A_1, \ldots, A_n$. Then m nodes labeled with $A_1, \ldots, A_n$ can be appended as children to the final node of this branch.

Let *PUHR+ tableaux* be trees built starting from the tree consisting of a single unlabeled node, by repeated application of the PUHR+ and Splitting+ rules.

There is an important connection between PUHR+ tableaux and positive hyperresolution:

Lemma 1. *Let C be a node label in a PUHR+ tableau for an input clause set $\mathcal{C}$. Then C is a subclause of some clause that can be generated from $\mathcal{C}$ by a finite number of PHR steps.*

Proof: (Sketch) Notice that every node of an infinite PUHR+ tableau also occurs in a finite one. For finite PUHR+ tableaux the proof of the lemma is straightforward by induction on the sequence of expansion steps by which the tableau has been generated: The initial case and splitting steps are trivial. For a PUHR+ step producing a clause C it can be shown that there is a corresponding PHR step that generates a clause of which C is a subclause.

Using the definition of PDC, we immediately get the following two corollaries:

Corollary 2. *All node labels in a PUHR+ tableau for an input clause set in PDC are decomposed.*

Corollary 3. *There are only finitely many different positive clauses that may appear as node labels in PUHR+ tableaux for some given input clause set in PDC.*

Because of corollary 2 the splitting rule may be applied without losing the model completeness (property 2) of PUHR tableaux (Section 2.1). Remember that the problem was that a splitting step corresponds to moving the universal quantifiers from the beginning of a clause to the literals, which does not always lead to an equivalent formula when applied to a non–range-restricted clause. But this transformation does lead to an equivalent formula when applied to a decomposed clause.

Because of corollary 3 every PUHR+ tableau construction process that avoids applying some PUHR+ step infinitely often terminates for input clause sets in PDC.

Open and *closed* branches and tableaux are defined for PUHR+ tableaux in the same way as for PUHR tableaux. The definition of saturation of branches in PUHR+ tableaux is slightly different from the corresponding one for PUHR

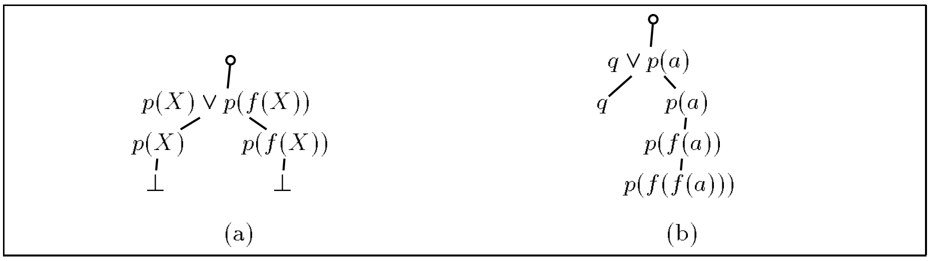

Fig. 4. Two PUHR tableaux for non-PDC clause sets.

tableaux, because variables can be taken into account: For a positive clause occurring as a node label in a branch we need not require that one of its atoms occurs as well, but only that it is subsumed by an atomic node label in the branch.

Properties 1 to 3 of PUHR tableaux (Section 2.1) also hold for PUHR+ tableaux, where of course the mentioned "set of atomic labels of an open branch" has to be replaced by the set of ground instances of these atomic labels.

We have now seen that both the FL method and some straightforward modification of the PUHR-tableau calculus are applicable to input clause sets in the class PDC. But there are also clause sets that can be handled only by one of the two approaches. Of course, they are not in the class PDC.

On the one hand, the clause set

$$\begin{array}{l} p(X) \vee p(f(X)) \\ p(f(X)) \rightarrow \bot \end{array}$$

cannot be handled by PUHR+ tableaux, but can be handled by the FL method: The empty clause is derived with two PHR steps. A saturated PUHR+ tableau for this clause set is in fact closed (Fig. 4a), but this is not a proof for the unsatisfiability of the clause set, since it uses an unsound splitting step for a non-decomposed clause.

On the other hand, the clause set

$$\begin{array}{l} q \vee p(a) \\ p(X) \rightarrow p(f(X)) \end{array}$$

cannot be handled by the FL method, but can be handled by PUHR and PUHR+ tableaux, if branches are selected for expansion in a fair way. The tableau in Fig. 4b has a finite saturated left branch which represents a model of the input clauses, although the right branch will eventually become infinite. The FL method generates the infinite set $\{q \vee p(f^i(a)) \mid i \in \mathbb{N}\}$ of clauses where no clause can be subsumed. So no model can be generated.

For each of these examples the fact that it works with (just) one of the approaches appears somewhat incidental. It is probably hard to describe large interesting classes of clause sets for which one of the approaches works and the other one does not.

Another difference in functionality of the two approaches is that PUHR tableaux are able to generate *all minimal* finite Herbrand models of a clause set, and—as noted above—a modified variant of PUHR tableaux is even able to generate exactly the minimal Herbrand models, if they are finite. In contrast to this, the FL method is only able to generate *some* Herbrand model, for which no minimality properties are known.

3.2 Efficiency

Since the FL method is only able to generate some model without any additional known properties, a fair comparison requires that PUHR tableaux are also only constructed as far as it is needed to find one model, i.e., one saturated branch, and that none of the modifications for concentrating on minimal models are needed. In the rest of this section a PUHR tableau will always be expanded at its leftmost open branch and a PUHR step will never be performed more than once in a branch. Similarly with the FL method a PHR step will never be performed more than once.

For comparing the efficiency of the two approaches simplified measures of complexity are used:

- For PUHR tableaux we count the number of tableau nodes that are generated before the first model is found or the unsatisfiability of the input clauses has been proven. This ignores the complexity of the expansion steps themselves.
- For the FL method we count the number of positive clauses that have been generated before the first selection step or before the empty clause has been derived. This ignores computations that take place after the selection step. However, experience shows that these computations are usually negligible compared to the computations before. It also ignores the complexity of the deduction steps themselves and especially of the subsumption steps, which might need considerably more time than the implicit *unit* subsumption in the splitting step of PUHR tableaux.

Nevertheless these measures should give a realistic impression for the complexity of the procedures.

For the sake of simplicity, the examples used for the comparison are propositional. There are highly efficient specialized procedures for deciding propositional problems, which probably beat both PUHR tableaux and the FL method. So one could argue that the examples are not in the typical application domain of the two model generators. However, it is straightforward to transform them into first-order examples with similar behavior.

Example 1.

$$\begin{array}{l} a_1 \vee b_1 \\ \vdots \\ a_n \vee b_n \\ a_n \rightarrow \bot \\ b_n \rightarrow \bot \end{array}$$

For this clause set the FL method generates the empty clause in two steps. It might also copy all the clauses from the input clause set to the pc-set. (These are trivial PHR steps.) So the complexity is between $O(1)$ and $O(n)$.

The PUHR tableau calculus might be lucky to choose the clause $a_n \vee b_n$ and split it immediately. Then both branches can be closed by the negative input clauses. The (more realistic) worst case is that a complete binary tree of height n with splitting of $a_i \vee b_i$ at level i is constructed before every branch is finally closed. So the complexity is between $O(1)$ and $O(2^n)$.

This advantage for the FL method comes from the fact that PUHR tableaux do repeated work for all branches, while FL avoids such a branching. Even the selection step, which can be seen as a "committed" variant of the splitting step, does not lead to branching. There is, however, a price to be paid for this, as we will see later with Example 2.

If the two negative rules are omitted, then the leftmost branch (of length $2n$) of any PUHR tableau represents a model. The FL method copies the n input clauses to the pc-set before it performs a selection step. So both approaches have complexity $O(n)$.

Example 2.

$$\begin{array}{l} a_1 \vee \ldots \vee a_n \\ a_1 \rightarrow b_1 \\ \vdots \\ a_n \rightarrow b_n \end{array}$$

For this clause set the PUHR tableau calculus performs one splitting step for the first clause. Then the leftmost branch with node label a_1 is extended by a node with label b_1. Now it is saturated and represents a model. The complexity is $O(1)$ or $O(n)$, depending on whether the $n - 1$ siblings of the node a_1 have actually been created already.

The FL method generates the 2^n positive clauses of the form $L_1 \vee \ldots \vee L_n$ where each L_i may be either a_i or b_i. So the complexity is $O(2^n)$.

If the additional clauses

$$\begin{array}{l} b_1 \rightarrow \bot \\ \vdots \\ b_n \rightarrow \bot \end{array}$$

are added, which makes the clause set unsatisfiable, then the complexity for PUHR tableaux remains $O(n)$, since all the n branches of the generated closed

PUHR tableau have constant length. With a sensible strategy for hyperresolution the complexity for the FL method drops to $O(n)$, whereas it remains $O(2^n)$ if all the 2^n clauses mentioned above are generated before the empty clause.

We learn the following heuristics from these and other examples:

- If a clause set is "easily" satisfiable, i.e., the "density" of models in the PUHR tableau search space is high, then PUHR tableaux typically find a model quickly without (much) branching. The FL method generates a large number of clauses in order to avoid branching, which is not (or hardly) done by PUHR tableaux anyway.
- If a clause set is "hard" to satisfy or even unsatisfiable, then the FL method can find a model or a contradiction faster because typically much subsumption can take place, which reduces the size and number of positive clauses. On the other hand, with PUHR tableaux many branches that are eventually closed have to be traversed before a model or (after the entire tableau) a contradiction is found.

The historical reason for this difference lies in the different application areas for which PUHR tableaux and hyperresolution have been developed. Satchmo has originally been developed for testing the satisfiability of integrity constraints in databases. For this application the desired and usual case is that a set of integrity constraints is satisfiable, i.e., that there is a possible database instance (a model) that satisfies the integrity constraints. The FL method uses full hyperresolution with subsumption, which has originally been developed for theorem proving. Here the normal and desired case is that some formula is a theorem, i.e., (the clausal form of) its negation is unsatisfiable.

Of the two fixpoint approaches for minimal model reasoning in disjunctive databases that are investigated by D. Seipel [13] one is similar to PUHR tableaux, and the other is similar to the FL method. Nevertheless, Seipel's efficiency considerations lead to a conclusion different to the one given above. The reason for this is that Seipel is interested in generating or representing *all* minimal models rather than a *single* one.

Finally, note that for Horn clauses, where neither branching nor non-unit hyperresolution occurs, PUHR tableaux and the FL method coincide.

4 Synthesis

In this section some guidelines for a combination of PUHR tableaux and the FL method are presented. PUHR tableaux and the FL method are in a sense complementary, since they have their strengths in different classes of applications. If it is known in advance from the application domain to which class a clause set belongs, i.e., which "density of models" one can expect, then simply one of the two approaches can be chosen. But sometimes one does not have this knowledge in advance. It is also possible to have a clause set of a "mixed nature", i.e., some part of the clause set is hard to satisfy while another (relatively independent) part is easy to satisfy, and these parts are not known in advance.

For such situations it is desirable to have a hybrid approach that adapts itself to the problem. Such a hybrid approach must support both branching and hyperresolution with non-unit electrons. The resulting calculus contains full PHR, subsumption and the selection rule. These rules now operate on tableau branches which are anyway similar to pc-sets. The splitting can become more flexible in that a clause of many literals need not necessarily be split into atoms but may be partitioned into subclauses in an arbitrary way. These non-unit subclauses can then be used as electrons in PHR steps.

Sometimes the user, i.e., the person or system from which an input clause set comes, can give hints, which disjunctions should be split and which ones should not. For the case where such additional information is not available, the following approach for controlling the calculus is suggested:

In addition to the tableau data structure that is maintained, there is a parameter for the maximal clause size (a positive integer). All positive clauses with a number of literals greater than this parameter are split into parts with a clause size that is at most the maximal clause size. Smaller positive clauses are used as electrons in PHR steps. The tableau to be constructed is traversed in a depth-first manner. Initially, the maximal clause size is 1, i.e., the hybrid approach behaves like the PUHR tableau calculus. According to the considerations in the previous section, this reflects the hope that a model can be found easily by PUHR tableaux. If this is really the case, then the hybrid approach finds a model as easily. If, however, it turns out that many branches have to be closed, the maximal clause size is incremented and the hybrid approach assumes a more and more FL-like behaviour. This reflects the fact that it has been recognized that the input clause set is not so easy to satisfy.

Of course it is possible to construct clause sets where the hybrid approach is especially inefficient. Let all models be concentrated in a part of the tableau that is traversed late, but let there be many models. Then the first (closed) branches of the tableau, where FL-like behavior would be better, are traversed with a PUHR-like behavior. When after a lot of backtracking the area of the models is reached, where the PUHR-like behavior would be better, the maximal clause size parameter has grown and the behavior is in fact more FL-like. A remedy to this problem could be to perform random permutation of literals in the clauses to be split in order to achieve a more uniform model density in the search.

5 Conclusion

The central contribution of this paper has been a comparison between two model generation methods: Satchmo by Manthey and Bry [11] with its formalization as PUHR tableaux by Bry and Yahya [2, 3] on the one hand, and the hyperresolution-based approach by Fermüller and Leitsch [5] on the other hand.

For this purpose, a minor extension of PUHR tableaux has been introduced, which is able to handle a class of clause sets similar to the one handled by Fermüller's and Leitsch's approach. So the two approaches do not differ much as far as their classes of solvable problems are concerned. The interesting difference is in the classes of clause sets that the two approaches can handle efficiently. This has suggested a combination of the approaches and has lead to guidelines for the combination.

It would be interesting to see if refinements of Satchmo that have been described in the literature (e.g., Satchmore [10] and *hyper tableaux* [1]) can still be combined with the Fermüller-Leitsch approach, or if similar refinements can be applied to the latter. The *hyper tableaux* calculus is a refinement of the PUHR tableaux calculus that performs ground instantiation before splitting steps at runtime on demand rather than in a preprocessing step. The considerations about PUHR+ tableaux above demonstrate that such an instantiation is never necessary for input clause sets in the class PDC. For such clause sets fairness is also not an issue in hyper tableaux.

Acknowledgments

The author thanks Thomas Brüggemann, François Bry, and Norbert Eisinger for helpful discussions and comments on a preliminary version of this paper. The support for the author by the Bayerischer Habilitations-Förderpreis is appreciated.

References

1. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In *Logics in Artificial Intelligence: European Workshop, JELIA '96*, Springer LNAI 1126, 1996.
2. F. Bry and A. Yahya. Minimal model generation with positive unit hyperresolution tableaux. In *5th Workshop on Theorem Proving with Tableaux and Related Methods*, Springer LNAI, 1996.
3. F. Bry and A. Yahya. Minimal model generation with positive unit hyperresolution tableaux. Research Report PMS-FB-1996-1, Institut für Informatik der Universität München, May 1996. (Full version of [2]).
4. R. Caferra and N. Zabel. Extending resolution for model construction. In *Logics in AI: European Workshop JELIA '90*, Springer LNAI 478, pages 153–169, 1991.
5. C. Fermüller and A. Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–203, 1996.
6. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1987.
7. K. Inoue and C. Sakama. A fixpoint characterization of abductive logic programs. *J. Logic Programming*, 27:107–136, 1996.
8. W. H. Joyner, Jr. *Automatic Theorem-Proving and the Decision Problem.* PhD thesis, Harvard University, Cambridge, Mass., USA, 1973.
9. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.
10. D. W. Loveland, D. W. Reed, and D. S. Wilson. SATCHMORE: SATCHMO with RElevancy. *Journal of Automated Reasoning*, 14:325–351, 1995.
11. R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In *9th Int. Conf. on Automated Deduction (CADE)*, Springer LNCS 310, pages 415–434, 1988.

12. J. A. Robinson. Automatic deduction with hyper-resolution. *Int. J. Computational Mathematics*, 1:227–234, 1965.
13. D. Seipel. *Efficient Reasoning in Disjunctive Deductive Databases.* Habilitation thesis, University of Tübingen, Tübingen, Germany, 1995.
14. J. Slaney. FINDER (Finite Domain Enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University Automated Reasoning Project, Canberra, 1992.
15. J. Slaney. SCOTT: A model-guided theorem prover. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 109–114, 1993.
16. R. Smullyan. *First-Order Logic.* Springer-Verlag, 1968.
17. T. Tammet. Using resolution for deciding solvable classes and building finite models. In *Baltic Computer Science: selected papers*, Springer LNCS 502, pages 33–64, 1991.

This article was processed using the LaTeX macro package with LLNCS style