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

Ludwig
Maximilians–
Universität
München

Positive Unit Hyper-Resolution Tableaux for Minimal Model Generation

François Bry and Adnan Yahya

http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-8, August 1997

Positive Unit Hyper-Resolution Tableaux for Minimal Model Generation

François Bry [1] Adnan Yahya [2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, D – 80538 München, Germany

[2] Electrical Engineering Department, Birzeit University, Birzeit, Palestine

Contact person: François Bry at the above address.
Phone: 49-89-21 78-22 10, Fax : 49-89-21 78-22 11
bry@informatik.uni-muenchen.de

Abstract

Minimal Herbrand models for clausal theories are useful in several areas of computer science, e.g. automated theorem proving, program verification, logic programming, databases, and artificial intelligence. In most cases, the conventional model generation algorithms are inappropriate because they generate nonminimal Herbrand models and can be inefficient. This article describes a novel approach for generating the minimal Herbrand models of sets of clauses. The approach builds upon *positive unit hyper-resolution (PUHR) tableaux,* that are in general smaller than conventional tableaux. PUHR tableaux formalize the approach initially introduced with the theorem prover SATCHMO. Two minimal model generation procedures are described. The first one expands PUHR tableaux depth-first relying on a *complement splitting* expansion rule and on a form of backtracking involving constraints. A Prolog implementation, named MM-SATCHMO, of this procedure is described. The second minimal model generation procedure performs a breadth-first, constrained expansion of PUHR (complement) tableaux. Both procedures are optimal in the sense that each minimal model is constructed only once, and the construction of nonminimal models is interrupted as soon as possible. They are complete in the following sense: The depth-first minimal model generation procedure computes all minimal Herbrand models of the considered clauses provided these models are all finite. The breadth-first minimal model generation procedure computes all finite minimal Herbrand models of the set of clauses under consideration.

1 Introduction:

Generating Herbrand models of clausal theories is useful in several areas of computer science. In automated theorem proving, models can assist in making conjectures, that can be later checked for provability with conventional provers. In automated theorem proving and program verification, model generation can also be applied to searching for counter-examples to conjectures. In both application areas, it is worthwhile and helpful to restrict model generation to minimal models.

The generation of minimal models is useful in logic programming and deductive databases for specifying their declarative semantics [14, 15], in some approaches to query answering [7, 12, 35, 34], for updating database facts and views [6, 9, 30, 3], in artifical intelligence for solving design synthesis and diagnosis problems [22, 25, 2], and in nonmonotonic reasoning [11] – see also [24]. Artificial intelligence production systems can be seen as minimal model generators for propositional or first-order logic Horn clauses.

The conventional tableaux methods [27, 8, 31, 32] are however inappropriate as model generation procedures because they often return redundant or nonminimal models [11, 21, 28, 16]. The a posteriori detection of redundant models is tedious and might be time consuming. Moreover, redundant models are a source of inefficiency because they blow up the search space. This article describes two procedures for generating the minimal Herbrand models of a set of first-order clauses. The proposed procedures are optimal in the sense that each minimal model is generated only once, and nonminimal models are rejected as soon as possible, in general before their complete construction. Measurements on an implementation in Prolog of one of the procedures, which is described in the paper, point to the efficiency of the approach.

Both prcedures are based on *positive unit hyper-resolution tableaux* (short *PUHR tableaux*), a (novel) formalization of an approach first introduced with the theorem prover SATCHMO [18, 19]. PUHR tableaux are ground and positive, more precisely their nodes consist of sets of ground atoms and disjunctions of ground atoms. They are expanded by means of only two rules, the *positive unit hyper-resolution* and the *splitting* (a simple version of β expansion [27, 8]) rules, from *range-restricted* clauses. Range-restriction is a syntactical property required in deductive database languages which is comparable to Skolemization: although requiring an extension of the language, it preserves models in a certain sense. The branching factor, the size of PUHR tableaux, and the size of the nodes of PUHR tableaux are in most cases significantly smaller than those of conventional tableaux. Positive unit hyper-resolution makes it possible not to blindly instantiate universally quantified variables. Instead, it combines in one step instantiations (or γ expansions [27, 8]) and splittings (or β expansion [27, 8]), thus reducing the depth of PUHR tableaux. Thanks to range-restrictedness full unification is not needed for computing positive unit hyper-resolvents. "Half-way unification" (or "merging") suffices.

The first minimal model generation procedure expands PUHR tableaux depth-first relying on a *complement splitting* expansion rule and on a form of backtracking involving constraints. *Complement splitting* rule (called "reduction" in [23] and "folding-down" in [13]) cuts out some branches leading to nonminimal models. Because PUHR tableaux are ground, complement splitting can be nicely and efficiently built into the method and into the SATCHMO programs. While discarding many nonminimal models, and preventing the generation of duplicate models, complement splitting is not always sufficient to reject all nonminimal models. In order to prune redundant models as soon as possible, a special depth first search strategy with extended backtracking is applied. The resulting depth-first minimal model generation procedure is sound in the sense that it generates only minimal Herbrand models, and complete in the sense that it returns all minimal Herbrand models of the input clauses, provided these minimal models are all finite. It is shown that this condition implies that there are finitely many minimal models. A variation, called MM-SATCHMO, of the SATCHMO program is given, which implements the depth-first minimal model generation procedure in Prolog.

The second minimal model generation procedure performs a breadth-first, possibly constrained expansion of PUHR (complement) tableaux. It is complete in the sense that it computes in finite time every finite minimal Herbrand model of the set of clauses under consideration.

The plan of the rest of this paper is as follows. Section 2 introduces terminology and notations, defines range-restricted clauses and PUHR tableaux, and recalls the SATCHMO implementation of PUHR tableaux. Section 3 is devoted to model generation using PUHR tableaux. Section 4 defines the depth-first and breadth-first minimal model generation procedures as a modified PUHR tableaux method and gives the Prolog implementation, called MM-SATCHMO, of the depth-first minimal model generation procedure.

The last chapter compares the proposed procedure with other approaches discussed in the literature, draws some conclusions, and points to possible directions for future research.

A preliminary version of this paper (without the proofs and Section 4.6) has been published in the Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods [4].

2 Preliminaries

2.1 Terminology and Notation

Throughout the paper usual terminology and notations are used, as in e.g. [27, 8]. When not explicitly otherwise stated, a first-order language $\mathcal{L}$ is implicitly assumed. It is also assumed that two special atoms $\top$ and $\bot$ are available, expressing respectively truth and falsity, i.e. $\top$ is satisfied in every interpretation, no interpretations satisfy $\bot$.

Every clause $C = L_1 \vee ... \vee L_k$ with negative literals $\{\neg A_1, ..., \neg A_n\}$ and positive literals $\{B_1, ..., B_m\}$ can be represented by a *clause in implication form*: $C' = A_1 \wedge .... \wedge A_n \rightarrow B_1 \vee ... \vee B_m$. $A_1 \wedge .... \wedge A_n$ is called the body of C', $B_1 \vee ... \vee B_m$ its head. If C contains no negative literals, $C' = \top \rightarrow B_1 \vee ... \vee B_m$. If C contains no positive literals, $C' = A_1 \wedge .... \wedge A_n \rightarrow \bot$.

A unifier σ of a conjunction of atoms $(A_1 \wedge .... \wedge A_n)$ and a sequence of atoms $(B_1, ..., B_n)$ (possibly with repeated atoms) is defined as a substitution σ such that $A_i\sigma = B_i\sigma$, for all $i = 1, ..., n$. If $(A_1 \wedge .... \wedge A_n)$ and $(B_1, ..., B_n)$ have a unifier, they are unifiable. Note that, since repeatitions in the sequence $(B_1, ..., B_n)$ are allowed, a conjunction $(A_1 \wedge .... \wedge A_n)$ might be unifiable with a sequence containig less than n (distinct) atoms. A unifier θ of $(A_1 \wedge .... \wedge A_n)$ and $(B_1, ..., B_n)$ is called a most general unifier (mgu) of $(A_1 \wedge .... \wedge A_n)$ and $(B_1, ..., B_n)$, if for each unifier σ of $(A_1 \wedge .... \wedge A_n)$ and $(B_1, ..., B_n)$, there exists a substitution γ such that $\sigma = \theta\gamma$.

An atom A is said to *subsume* an atom B (a disjunction of atoms $B_1 \vee ... \vee B_n$, resp.) if there exists a substitution σ such that $A\sigma = B$ ($A\sigma = B_i$ for some $i \in \{1, ..., n\}$, resp.).

An interpretation of $\mathcal{L}$ will be denoted as a pair $(\mathcal{D}, m)$ where the nonempty set $\mathcal{D}$ is the universe (or domain) and m is the mapping interpreting the symbols and expressions of the language.

The *universal closure* of a clause C is $\forall x_1 ... \forall x_n C$, where $x_1, ..., x_n$ are the variables occurring in C. A clause (resp. a set of clauses) is said to be *satisfied* by an interpretation when the universal closure of the clause (resp. the set of the universal closures of the clauses) is satisfied by this interpretation. A clause (resp. a set of clauses) is said to be *satisfiable* if it has at least one interpretation in which it is satisfied. A clause (resp. a set of clauses) is said to be *finitely satisfiable* if it is satisfied by an interpretation with a finite domain.

A term or formula in which no variable occurs is said to be *ground.* If $\mathcal{A}$ is a set of ground atoms, $H(\mathcal{A})$ denotes the Herbrand interpretation which satisfies a ground atom B if and only if $B \in \mathcal{A}$. A Herbrand interpretation $H(\mathcal{A})$ is said to be *finitely representable* if $\mathcal{A}$ is finite. Since confusions can be avoided from the context, a set of formulas having a finitely representable Herbrand model will be said to be *finitely representable.* Note that finite representability (of sets of formulas) and finite satisfiability are two distinct properties.

The subset relationship $\subseteq$ over sets of ground atoms induces an order $\leq$ on Herbrand interpretations: given two sets $\mathcal{A}_1$ and $\mathcal{A}_2$ of ground atoms, $H(\mathcal{A}_1) \leq H(\mathcal{A}_2)$ if and only if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. If $\mathcal{S}$ is a set of clauses, $\leq$ induces an order on Herbrand models of $\mathcal{S}$. A Herbrand model $H(\mathcal{A})$ of $\mathcal{S}$ is said to be a minimal Herbrand model of $\mathcal{S}$ if it is minimal for $\leq$, i.e. for every

Herbrand model $H(\mathcal{A}')$ of $\mathcal{S}$, if $H(\mathcal{A}') \leq H(\mathcal{A})$, then $\mathcal{A}' = \mathcal{A}$.

If $\mathcal{E}$ is a set of formulas, $Atoms(\mathcal{E})$ denotes the set of atoms (i.e. positive unit clauses) that are elements of $\mathcal{E}$.

Variables are denoted by x and y with or without subscripts, constants by a, b, c or d, predicate symbols by D, P, Q, and R, and function symbols by f.

In this paper a tableau method and a minimal model generation procedure for *clausal* theories are defined, i.e. it is assumed that existential quantifications have been removed through Skolemization.

2.2 Range Restriction

Definition 1 (Range restricted clause) *A clause (resp. a clause in implication form) is said to be range restricted if every variable occurring in a positive (resp. head) literal also appears in a negative (resp. body) literal.*

Clearly, a range restricted clause in implication form is ground if its body is ground, e.g. if it is $\top$. A transformation is first defined, which associates a set $RR(\mathcal{S})$ of range restricted clauses in implication form with every set $\mathcal{S}$ of clauses in implication form.

Definition 2 (Range restriction transformation) *Let $\mathcal{L}'$ be an extension of the language $\mathcal{L}$ with a unary predicate D (not belonging to $\mathcal{L}$).*

For every $\mathcal{L}$-clause $C = A_1 \wedge ... \wedge A_n \rightarrow B_1 \vee ... \vee B_m$, let $RR(C)$ be the following $\mathcal{L}'$-clause:

$$RR(C) := \begin{cases} C \text{ if } C \text{ is range restricted;} \\ D(x_1) \wedge ... \wedge D(x_k) \wedge A_1 \wedge ... \wedge A_n \rightarrow B_1 \vee ... \vee B_m \text{ otherwise,} \\ \text{where } x_1, ..., x_k \text{ are the variables occurring in the } B_i\text{s and in} \\ \text{none of the } A_j\text{s.} \end{cases}$$

Let $\mathcal{S}$ be a set of $\mathcal{L}$-clauses. For a term t distinct from a variable occurring in $\mathcal{S}$, let C_t be the $\mathcal{L}'$-clause:

$$C_t := \begin{cases} D(x_1) \wedge ... \wedge D(x_k) \rightarrow D(t) \text{ if the variables } x_1, ..., x_k \text{ occur in } t; \\ \top \rightarrow D(t) \text{ if no variables occur in } t. \end{cases}$$

Let τ be the set of nonvariable terms occurring in $\mathcal{S}$. Let $\mathcal{S}'$ be the following set of $\mathcal{L}'$-clauses:

$$\mathcal{S}' := \begin{cases} \{C_t \mid t \in \tau\} \text{ if } \tau \text{ contains a constant;} \\ \{C_a\} \cup \{C_t \mid t \in \tau\} \text{ otherwise, for some constant } a. \end{cases}$$

$RR(\mathcal{S}) := \{RR(C) \mid C \in \mathcal{S}\} \cup \mathcal{S}'$ is the range restriction of $\mathcal{S}$.

Note that by construction the clauses in $RR(\mathcal{S})$ are range restricted and that $RR(\mathcal{S})$ is finite if $\mathcal{S}$ is finite. Strictly speaking, the range restriction transformation does not preserve models because it extends the language $\mathcal{L}$ with the unary predicate D.

Example 1

1. If $\mathcal{S} = \{\top \rightarrow P(f(x))\}$, then $RR(\mathcal{S}) = \{D(x) \rightarrow P(f(x))\ ,\ \top \rightarrow D(a)\ ,\ D(x) \rightarrow D(f(x))\}$ where, in the first clause, $D(x) \wedge \top$ is simplified into $D(x)$.

2. If $\mathcal{S} = \{P(x,y) \rightarrow P(f(x),y)\}$, then $RR(\mathcal{S}) = \{P(x,y) \rightarrow P(f(x),y)\ ,\ \top \rightarrow D(a)\ ,\ D(x) \rightarrow D(f(x))\}$.

The following theorem shows that the range restriction transformation preserves models in a certain sense, similar to the way Skolemization does.

Theorem 3 *Let $\mathcal{S}$ be a set of clauses in a language $\mathcal{L}$ (with no other function symbols than those occurring in $\mathcal{S}$ except possibly the constant a). Let $RR(\mathcal{S})$ be the range restriction of $\mathcal{S}$ (in an extension $\mathcal{L}'$ of $\mathcal{L}$ with a unary predicate D).*

1. *If $(\mathcal{D}, m)$ is a model of $\mathcal{S}$ and if m' is the mapping over $\mathcal{L}'$ defined as follows:*
$$m'(s) := \begin{cases} m(s) & \text{if } s \neq D, \\ \mathcal{D} & \text{if } s = D. \end{cases}$$
then $(\mathcal{D}, m')$ is a model of $RR(\mathcal{S})$.

2. *If $(\mathcal{D}, m')$ is a model of $RR(\mathcal{S})$, then $(\mathcal{D}, m'|_{\mathcal{L}})$ is a model of $\mathcal{S}$, where $m'|_{\mathcal{L}}$ denotes the restriction of m' to $\mathcal{L}$.*

Proof: Point 1 follows immediately from Definition 2. For point 2 the non-emptiness of $\mathcal{S}'$ (cf. Definition 2) is necessary, because the clauses $RR(C)$ are satisfied over any interpretation mapping the added unary predicate D to the empty set. ∎

This result means that range restrictedness can be seen as just a special syntactic form rather than a real restriction – from a theoretical point of view. For practical purposes, on the other hand, range restrictedness does make a difference. In the context of PUHR tableaux, the range restriction transformation induces an enumeration of the ground terms, making the γ expansion rule of conventional tableaux [27, 8] superfluous. Thus, if the procedures presented in this paper are applied to a set $RR(\mathcal{S})$ obtained from $\mathcal{S}$ by the transformation above, then the newly introduced atoms with predicate D have basically the same effect as an instantiation – or γ – rule for the clauses of the original set $\mathcal{S}$.

When applied in a refutation procedure, instantiation is often a source of inefficiency. Note, however, that this is not the case for model generation. In contrast to refutation, model generation requires instantiation anyway, indeed, for Herbrand models are characterized as sets of *ground* atoms.

Definition 4 (Positive unit hyperresolution) *Let $C = A_1 \wedge ... \wedge A_n \rightarrow E_1 \vee ... \vee E_m$ be a clause in implication form, $B_1, ..., B_n$ be n (not necessarily distinct) atoms such that $(A_1 \wedge ... \wedge A_n)$ unifies with $(B_1, ..., B_n)$. If σ is a most general unifier of $(A_1 \wedge ... \wedge A_n)$ and $(B_1, ..., B_n)$, then $(E_1 \vee ... \vee E_m)\sigma$ is a positive unit hyper-resolvent of C and $B_1, ..., B_n$.*

Lemma 5 *The positive unit hyper-resolvent of a range restricted clause in implication form and ground atoms is a ground atom or a disjunction of ground atoms.*

Proof: Immediate. ∎

Note that no occur-checks need to be performed for computing the positive unit hyper-resolvent of a range restricted clause in implication form and ground atoms. Indeed, half-way unification (or matching) suffices in computing a positive unit hyper-resolvent of a range restricted clause in implication form and of ground atoms.

In the next section, positive unit hyper-resolution tableaux are defined for range restricted clauses. This is not a significant restriction, for there is a transformation of any set of clauses into a set of range-restricted clauses which preserves models in the sense of Theorem 3. Note also that most database and artificial intelligence applications naturally yield range restricted specifications.

2.3 Positive Unit Hyper-Resolution Tableaux

Starting from the set $\{\top\}$, the PUHR tableaux method expands a tree – or positive unit hyper-resolution (PUHR) tableau – for a set $\mathcal{S}$ of range restricted clauses in implication form by applying the following expansion rules that are defined with respect to $\mathcal{S}$. The nodes of a PUHR tableau are sets of ground atoms or disjunctions of ground atoms.

Definition 6 (PUHR tableaux expansion rules) *Let $\mathcal{S}$ be a set of clauses in implication form.*

- *Positive unit hyper-resolution (PUHR) rule:*

$$\frac{\begin{array}{c} B_1 \\ \vdots \\ B_n \end{array}}{E\sigma}$$

where σ is a most general unifier of the body of a clause $(A_1 \wedge ... \wedge A_n \rightarrow E) \in \mathcal{S}$ *and of* $(B_1, ..., B_n)$.

- *Splitting rule:*

$$\frac{E_1 \vee E_2}{E_1 \mid E_2}$$

In the following definition, the splitting rule is applied to *ground* disjunctions.

Definition 7 (PUHR tableaux) *Positive unit hyper-resolution (PUHR) tableaux for a set $\mathcal{S}$ of clauses in implication form are trees whose nodes are sets of ground atoms and disjunctions of ground atoms. They are inductively defined as follows:*

1. $\{\top\}$ *is a positive unit hyper-resolution tableau for $\mathcal{S}$.*
2. *If T is a positive unit hyper-resolution tableau for $\mathcal{S}$, if L is a leaf of T such that an application of the PUHR rule (resp. splitting rule) to formulas in L yields a formula E (resp. two formulas E_1 and E_2) not subsumed by an atom in L, then the tree T' obtained from T by adding the node $L \cup \{E\}$ (resp. the two nodes $L \cup \{E_1\}$ and $L \cup \{E_2\}$) as successor(s) to L is a positive unit hyper-resolution tableau for $\mathcal{S}$.*

A branch of a positive unit hyper-resolution tableau is said to be closed, if it includes a node containing the atom $\perp$. A positive unit hyper-resolution tableau is said to be closed if all its branches are closed. A branch (resp. tableau) which is not closed is said to be open.

A positive unit hyper-resolution tableau T for $\mathcal{S}$ is said to be satisfiable if the union of $\mathcal{S}$ with the nodes of a branch of T is satisfiable.

Note that if $\mathcal{P}$ is a path from the root to a node N of a PUHR tableaux, then by Definition 7, $N = \cup\mathcal{P}$.

Convention. If N_1 and $N2$ are the nodes of a PUHR tableau T containing respectively E_1 and E_2 and resulting from an application of the splitting rule to $E_1 \vee E_2$, it is assumed in the sequel that the PUHR tableau T is ordered such that N_1 is the left sibling of E_2.

Example 2 Figure 1 gives a PUHR tableau for the following set of clauses in implication form:

$$\begin{array}{ll} \top \rightarrow P(a) \vee Q(b) & P(b) \rightarrow \perp \\ P(x) \rightarrow P(f(x)) \vee Q(f(x)) & P(f(x)) \rightarrow \perp \\ Q(x) \rightarrow P(x) \vee R(x) & P(x) \wedge Q(f(x)) \rightarrow \perp \end{array}$$

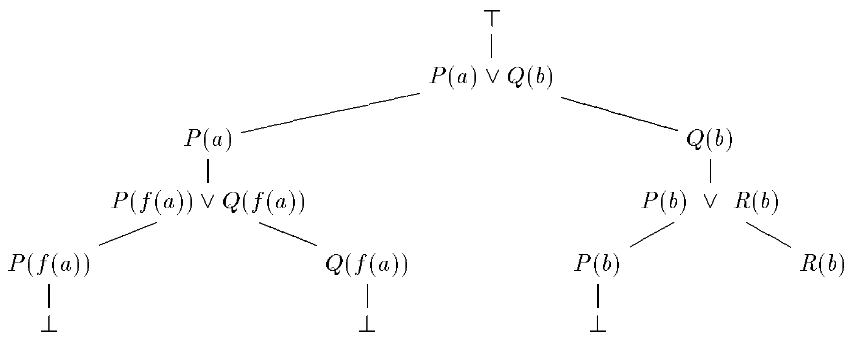


Figure 1: A PUHR tableau for the set of clauses of Example 2.

For the sake of readability, the nodes of the tree of Figure 1 are labeled with the ground atoms or disjunctions of ground atoms added at these nodes. We recall that by Definition 7 the nodes of a PUHR tableau are sets of ground atoms and disjunctions of ground atoms.

By Lemma 5 the nodes of a positive unit hyper-resolution tableau for a set of range restricted clauses are sets of ground atoms and disjunctions of ground atoms. Note that sets of clauses for which PUHR tableaux are defined may be infinite. According to Definition 6 clauses whose heads are $\perp$ only contribute to close branches. Since negative formulas do not explicitly occur in PUHR tableaux, closure is simply detected by the presence of $\perp$, which is simpler than checking for atomic closure [8].

Definition 8 *Let $\mathcal{S}$ be a set of range-restricted clauses in implication form and $\mathcal{A}$ a set of ground atoms and disjunctions of ground atoms. $\mathcal{A}$ is said to be saturated with respect to $\mathcal{S}$ for the positive unit hyper-resolution and splitting expansion rules when the following properties hold:*

1. *if $(A_1 \wedge ... \wedge A_n \rightarrow E) \in \mathcal{S}$, $B_1 \in \mathcal{A}$, ..., and $B_n \in \mathcal{A}$, and $(A_1 \wedge ... \wedge A_n)$ and $(B_1, ..., B_n)$ are unifiable, then $E\sigma \in \mathcal{A}$ for a most general unifier σ of $(A_1 \wedge ... \wedge A_n)$ and $(B_1, ..., B_n)$.*
2. *If $(E_1 \vee E_2) \in \mathcal{A}$, then $E_1 \in \mathcal{A}$ or $E_2 \in \mathcal{A}$.*

Note that if $\mathcal{B}$ is an open or a closed branch of a PUHR tableau, then $\cup\mathcal{B}$ is not necessarily saturated. As well, if $\cup\mathcal{B}$ is saturated, then $\mathcal{B}$ is neither necessarily open, nor necessarily closed.

Lemma 9 *The application of an expansion rule to a satisfiable PUHR tableau results in a satisfiable PUHR tableau.*

Proof: If M is a model of a set $\mathcal{F}$ of clauses, atoms and disjunctions, and if E is a positive unit hyper-resolvent of elements of $\mathcal{F}$, then $M \models E$. If M is a model of $\mathcal{F}$ and $E_1 \vee E_2 \in \mathcal{F}$, then $M \models E_1$ or $M \models E_2$. ∎

Theorem 10 (Refutation soundness) *Let $\mathcal{S}$ be a set of range-restricted clauses in implication form. If there exists a closed PUHR tableau for $\mathcal{S}$, then $\mathcal{S}$ is unsatisfiable.*

Proof: Assume $\mathcal{S}$ is satisfiable. By Lemma 9 there are no closed PUHR tableaux for $\mathcal{S}$. ■

Definition 11 *A PUHR tableau is said to be fair, if the union of the nodes of each of its open branches is saturated for the expansion rules.*

Informally, a PUHR tableaux is fair if along each of its open branches, each possible application of an expansion rule is performed at least once.

If $\mathcal{B}$ is a branch of a tableau, then $Atoms(\cup\mathcal{B})$ denotes the set of atoms (i.e. positive unit clauses) that are elements of some nodes in $\mathcal{B}$. In the sequel, $Atoms(\mathcal{E})$ will always be refered to in cases where all atoms in $\mathcal{E}$ are *ground*. Recall that if $Atoms(\mathcal{E})$ is a set of ground atoms, it characterizes the Herbrand interpretation $H(Atoms(\mathcal{E}))$.

Lemma 12 *Let $\mathcal{S}$ be a set of range-restricted clauses in implication form and $\mathcal{E}$ be a set of ground atoms and disjunctions of ground atoms. If $\mathcal{S} \cup \mathcal{E}$ is saturated for the expansion rules with respect to $\mathcal{S}$ and if $\mathcal{E}$ does not contain $\perp$, then $H(Atoms(\mathcal{E}))$ is a model of $\mathcal{S}$.*

Proof: Immediate. ■

Theorem 13 (Refutation completeness) *Let $\mathcal{S}$ be a set of range-restricted clauses in implication form. If $\mathcal{S}$ is unsatisfiable, then every fair positive unit hyper-resolution tableau for $\mathcal{S}$ is closed.*

Proof: Let T be an open fair PUHR tableau for $\mathcal{S}$, and $\mathcal{B}$ an open branch of T. Since T is fair, then $\cup\mathcal{B}$ is saturated for the expansion rules. By Lemma 12 $H(Atoms(\cup\mathcal{B}))$ is a model of $\mathcal{S}$. Hence $\mathcal{S}$ is satisfiable. ■

PUHR tableaux are defined for sets of range restricted clauses. Combined with the PUHR expansion rule of Definition 6, the range restriction transformation induces an enumeration of the ground terms, as observed in [17].

2.4 Implementation in Prolog

The Prolog program of Figure 2 expands fair PUHR tableaux for sets of range-restricted clauses in implication form under a depth-first search strategy. The tableaux expanded by this program are strict [8] and subsumption-free. Strictness means that no application of an expansion rule is performed more than once to given clauses, atoms, or disjunctions. Subsumption-freeness means that only ground disjunctions that are not subsumed by previously generated atoms or disjunctions can be split.

```
satisfiable :-
     findall(Clause, violated_instance(Clause), Set),
     not (Set = []), !,
     satisfy_all(Set),
     satisfiable.
satisfiable.

violated_instance(B ---> H) :-
     (B ---> H), B, not H.

satisfy_all([]).
satisfy_all([_B ---> H | Tail]) :-
     H, !, satisfy_all(Tail).
satisfy_all([_B ---> H | Tail]) :-
     satisfy(H), satisfy_all(Tail).

satisfy(E) :-
    component(Atom, E), not (Atom = false),
    assume(Atom).

component(Atom, (Atom ; _Rest)).
component(Atom, (_ ; Rest)) :-
     !, component(Atom, Rest).
component(Atom, Atom).

assume(Atom) :-
     asserta(Atom).
assume(Atom) :-
     once(retract(Atom)),
     fail.
```

Figure 2: The SATCHMO program.

Backtracking over `satisfiable` returns Herbrand models $H(\mathcal{M})$. The ground atoms of $\mathcal{M}$ are inserted into the Prolog database by the predicate `assume`. On backtracking, they are removed. A clause $A_1 \wedge ... \wedge A_n \rightarrow B_1 \vee ... \vee B_m$ is represented in the Prolog database as

```
A1, ..., An ---> B1 ; ... ; Bm,
```

where `--->` is declared as an infix binary predicate. $\perp$ is represented as `false`, $\top$ as the built-in predicate `true`, which is always satisfied.

Fairness is ensured by the call to the all-solutions built-in predicate `findall`. The predicate `component` on backtracking successively returns the atoms of a disjunction. The predicate `satisfy` on backtracking successively returns the components of a disjunction that are not subsumed by atoms previoulsy inserted into the Prolog database. For each ground instance `_B ---> H` of a clause returned by the the call

```
findall(Clause, violated_instance(Clause), Set)
```

the predicate `satisfy_all` selects an atom in the head `H` and asserts it in the Prolog database. On backtracking, the different ways to satisfy the head `H` of each ground instance `_B ---> H` returned by the call to `findall` are considered.

The program of Figure 2, called SATCHMO, as well as variations of it have been first published in [18, 19]. In these articles, the programs are explained in more detail and performance on benchmark examples is reported. The PUHR tableaux introduced in Section 2.3 are a formalization of the principle of the SATCHMO programs. This is, to the best of our knowledge, the first formalization of the SATCHMO approach to theorem proving.

It is worth pointing out that `satisfy_all` is a simple and straightforward implementation which, in some cases, has drawbacks. Consider for example the following Prolog representations $\mathcal{R}_1$ and $\mathcal{R}_2$ of the same set of clauses:

$\mathcal{R}_1$:	$\mathcal{R}_2$:
`true ---> p(a)`	`true ---> p(b) ; p(a)`
`true ---> p(b) ; p(a)`	`true ---> p(a)`

Applied to $\mathcal{R}_1$, the call to

```
findall(Clause, violated_instance(Clause), Set),
```

instantiates `Set` with the list `[(true ---> p(a)),(true ---> p(b);p(a))]`. Then the call to `satisfy_all` first asserts `p(a)` into the Prolog database so as to satisfy the head of `true ---> p(a)`. Since now `p(b) ; p(a)` is satisfied, no further actions are taken, as specified by the second clause of `satisfy_all`. If in contrast $\mathcal{R}_2$ is considered, the call to

```
findall(Clause, violated_instance(Clause), Set)
```

binds `Set` to the list:

```
[(true ---> p(b) ; p(a)) (p(a), true ---> p(a))]
```

The call to `satisfy_all` now satisfies first `p(b) ; p(a)`, then `p(a)`. That is `p(b)` is first asserted, then `p(a)`. On backtracking, `p(a)` only is asserted.

Such a behaviour depending on the order of the clauses in Prolog can be avoided with a more sophisticated implementation of `satisfy_all` which satisfies the considered set of heads of ground clauses by a *minimal* set of atoms. Since such a refined implementation of `satisfy_all` is not needed for the purpose of this report, it is not given here.

3 Model Generation with PUHR Tableaux

In the previous section, PUHR tableaux were considered from the angle of refutation. In this section, their properties with respect to model generation are investigated.

Theorem 14 (Model soundness) *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form and T a fair PUHR tableau for $\mathcal{S}$. If $\mathcal{B}$ is an open branch of T, then $H(Atoms(\cup\mathcal{B}))$ is a model of $\mathcal{S}$.*

Proof: Fairness ensures saturation with respect to the expansion rules. Theorem 14 follows from Lemma 12. ∎

Theorem 15 *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form, T be a PUHR tableau for $\mathcal{S}$, and $\mathcal{M}$ a set of ground atoms. If $H(\mathcal{M})$ is a model of $\mathcal{S}$, then there is a branch $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) \subseteq \mathcal{M}$.*

Proof: Let $\mathcal{B}_0, ..., \mathcal{B}_i, ...$ be an enumeration of the branches of T, whose atoms are not in $\mathcal{M}$. For each $i \in \mathrm{N}$ let A_i be an atom of the branch $\mathcal{B}_i$ which is not in $\mathcal{M}$. Let $\mathcal{S}' = \mathcal{S} \cup \{A_i \rightarrow \perp \ : i \in \mathrm{N}\}$. By definition of $\mathcal{S}'$, since no A_i is in $\mathcal{M}$, $H(\mathcal{M})$ is also a model of $\mathcal{S}'$. Furthermore T can be extended into a positive unit hyper-resolution tableau T' of $\mathcal{S}'$ by adding $\perp$ to the successor nodes of those nodes of T that contain some A_i. Let $\mathcal{B}'_i$ denote such an extension of the branch $\mathcal{B}_i$ in T'. By Theorem 10, T' has an open branch, say $\mathcal{B}$. Since $\mathcal{B}$ is open, it is none of the $\mathcal{B}'_i$. Since all clauses of $\mathcal{S}$, whose heads are $\perp$, are also in $\mathcal{S}'$, $\mathcal{B}$ is also an open branch of T. $\mathcal{B}$ is none of the $\mathcal{B}_i$ because otherwise, by definition of T', it would be one of the $\mathcal{B}'_i$. By definition of the $\mathcal{B}_i$s $Atoms(\cup\mathcal{B}) \subseteq \mathcal{M}$. ∎

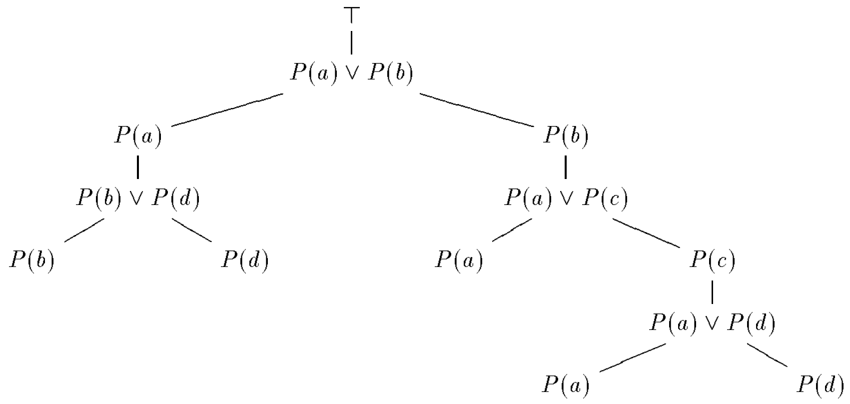


Figure 3: A PUHR tableau for Example 3 with nonminimal and duplicate models.

Corollary 16 (Minimal model completeness) *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form, T be a fair positive unit hyper-resolution tableau for $\mathcal{S}$, and $\mathcal{M}$ a set of ground atoms. If $H(\mathcal{M})$ is a minimal model of $\mathcal{S}$, then there is a branch $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) = \mathcal{M}$.*

Proof: By Theorem 15, there is a branch $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) \subseteq \mathcal{M}$. Since T is fair, by Theorem 14 $H(Atoms(\cup\mathcal{B}))$ is a model of $\mathcal{S}$. Since $H(\mathcal{M})$ is a minimal model of $\mathcal{S}$, $Atoms(\cup\mathcal{B}) = \mathcal{M}$. ∎

The following example demonstrates that a plain PUHR tableau can generate both, nonminimal and duplicate models.

Example 3 Let $\mathcal{S}$ be the following set of clauses:

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) & \qquad P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & \qquad P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Figure 3 is a PUHR tableau for $\mathcal{S}$. The minimal model $H(\{P(a), P(b)\})$ of $\mathcal{S}$ is generated twice, at the leftmost branch and at the third branch from the left of the PUHR tableau. The fourth branch from the left of the PUHR tableau generates the nonminimal model $H(\{P(a), P(b), P(c)\})$. Note that the PUHR tableau returns among others all minimal models of $\mathcal{S}$, i.e. $H(\{P(a), P(b)\})$, $H(\{P(a), P(d)\})$, and $H(\{P(b), P(c), P(d)\})$.

Corollary 16 is established, though in a different context, in [5] and mentioned without proof in [12]. As the following counter-example shows, fairness is necessary in Corollary 16, although not in Theorem 15.

Example 4 With the theory $\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow P(f(x)) \vee P(b), P(a) \rightarrow P(b)\}$ consistently expanding on the second clause will not allow the generation of the (only) minimal model $H(\{P(a), P(b)\})$ of $\mathcal{S}$.

4 Minimal Model Generation

By Corollary 16 fair PUHR tableaux generate all minimal models. However, they often also generate duplicate and/or nonminimal models, as e.g. in Example 3 above. A naive approach to minimal model generation consists in first expanding (fair) PUHR tableaux, and later pruning them from redundant branches. In this section a more efficient approach is described which consists in a depth-first expansion of PUHR tableaux combined with an extended backtracking which prunes the search space from redundant branches as soon as possible. Under certain finiteness conditions, this depth-first minimal model generation procedure is complete. However, it is inappropriate if some minimal models are infinite. The generation of minimal models based on breadth-first expansion of (fair) PUHR tableaux is finally discussed.

4.1 Finiteness Properties

Theorem 17 *Let $\mathcal{S}$ be a set of formulas. If $\mathcal{S}$ has a finitely representable Herbrand model it also has a finite model.*

Proof: Let $(\mathcal{D}, m)$ be a finitely representable Herbrand model of $\mathcal{S}$, and $\mathcal{A}$ be the set of ground atoms that are satisfied in $(\mathcal{D}, m)$. A finite model of $\mathcal{S}$ is built by identifying the elements of the universe $\mathcal{D}$ over which no terms occurring in $\mathcal{A}$ are mapped. Formally, let $\sim$ be the equivalence relation over $\mathcal{D}$ defined by: $d_1 \sim d_2$ if and only if $d_1 = d_2$ or for all $R(t_1, ..., t_n) \in \mathcal{A}$ and for all $i = 1, ..., n$, $m(t_i) \neq d_1$ and $m(t_i) \neq d_2$. Let f be the mapping of an element of $\mathcal{D}$ to its equivalence class for $\sim$ in $\mathcal{D}/\sim$. Let $\mathcal{D}' = \mathcal{D}/\sim$ and $m' = f \circ m$. Since $\mathcal{A}$ is finite, $\mathcal{D}/\sim$ is finite. By definition of $\mathcal{D}'$ and m', a ground atom is satisfied in $(\mathcal{D}', m')$ if and only if it is satisfied in $(\mathcal{D}, m)$. Since $(\mathcal{D}, m) \models \mathcal{S}$, it follows that $(\mathcal{D}', m') \models \mathcal{S}$. ■

The following result relates the finiteness of the set of minimal models to the finite representability of the minimal models. Let us call *finitary* a set of clauses, whose minimal Herbrand models are all finitely representable.

Theorem 18 *Let $\mathcal{S}$ be a set of clauses. If $\mathcal{S}$ is finitary, then $\mathcal{S}$ has finitely many minimal Herbrand models.*

Proof: Let $\mathcal{S}$ be a set of clauses with an infinite number of finitely representable minimal Herbrand models. Let $H(\mathcal{A}_0), ..., H(\mathcal{A}_n), ...$ be an enumeration of all finitely representable minimal Herbrand models of $\mathcal{S}$, such that the $\mathcal{A}_i$s are pairwise distinct. If $\mathcal{A}$ is a finite set of atoms $\{A_1, ..., A_k\}$, let

$Neg(\mathcal{A})$ denote the (singleton) set of clauses $\{A_1 \wedge ... \wedge A_k \rightarrow \perp\}$. For every $n \in \mathrm{N}$, let $\mathcal{S}^n = \mathcal{S} \cup Neg(\mathcal{A}_0) \cup ... \cup Neg(\mathcal{A}_n)$. Since all $\mathcal{A}_i$s are finite, the $\mathcal{S}^n$s are also finite. Each $\mathcal{S}^n$ is satisfiable because, by definition, $H(\mathcal{A}_{n+1})$ is a model of $\mathcal{S}^n$. Let $\mathcal{S}^\omega = \cup\{\mathcal{S}^n : n \in \mathrm{N}\}$. Since every $\mathcal{S}^n$ is satisfiable, every finite subset of $\mathcal{S}^\omega$ is satisfiable. By the compactness theorem, $\mathcal{S}^\omega$ is therefore satisfiable. Since $\mathcal{S}^\omega$ is a set of clauses, it has a Herbrand model, and therefore also some minimal Herbrand model $H(\mathcal{M})$. By definition of $\mathcal{S}^\omega$, $H(\mathcal{M})$ is none of the finitely representable models $H(\mathcal{A}_n)$. Therefore $\mathcal{M}$ is infinite. ∎

Although finite representability (of a set of formulas) is a stronger property than finite satisfiability, we conjecture that it is semi-decidable like finite satisfiability. We also conjecture that the finitary property is semi-decidable.

Let $\mathcal{S}$ be a set of clauses whose minimal Herbrand models are all finitely representable. By Theorem 18 a PUHR tableau for $\mathcal{S}$ pruned from those branches corresponding to nonminimal models is finite. Note, however, that a finitary theory may have inifinite nonminimal Herbrand models, as is shown by Example 5 below.

In many applications, the finite representability of the minimal Herbrand models is often assumed. This is the case in particular of disjunctive databases [15] and of some forms of nonmonotonic reasoning [25, 24, 11]. Thus, from the viewpoint of applications, Theorem 18 seems to be an important result.

4.2 Complement Splitting

If $C = A_1 \vee ... \vee A_n$ is an atom or a disjunction of atoms, let $Neg(C)$ denote the finite set of clauses in implication form $Neg(C) := \{A_1 \rightarrow \perp, ..., A_n \rightarrow \perp\}$.

Definition 19 (Complement splitting rule)

$$\frac{E_1 \vee E_2}{\begin{array}{l|r} E_1 & E_2 \\ Neg(E_2) & \end{array}}$$

Like the splitting rule, the complement splitting rule (already mentioned in [19], called reduction in [23] and folding-down in [13]) is applied in the following definitions to *ground* disjunctions. Tableaux expanded with the positive unit hyper-resolution and the complement splitting rules are defined inductively, similarly as in Definition 7. Let us call such tableaux *PUHR complement tableaux.* Note that nodes of PUHR complement tableaux are sets of ground atoms, disjunctions of ground atoms, and ground implications of the form $A \rightarrow \perp$.

Definition 20 (PUHR complement tableaux) *Positive unit hyper-resolution (PUHR) complement tableaux for a set $\mathcal{S}$ of clauses in implication form are trees whose nodes are sets of ground atoms, disjunctions of ground atoms, and ground implications of the form $A \rightarrow \bot$. They are inductively defined as follows:*

1. *$\{\top\}$ is a positive unit hyper-resolution complement tableau for $\mathcal{S}$.*

2. *If T is a positive unit hyper-resolution complement tableau for $\mathcal{S}$, if L is a leaf of T such that an application of the PUHR rule (resp. complement splitting rule) to formulas in L yields a formula E (resp. two sets of formulas $\{E_1, Neg(E_2)\}$ and $\{E_2\}$), then the tree T' obtained from T by adding the node $L \cup \{E\}$ (resp. the two nodes $L \cup \{E_1, Neg(E_2)\}$ and $L \cup \{E_2\}$) as successor(s) to L is a positive unit hyper-resolution complement tableau for $\mathcal{S}$.*

For PUHR complement tableaux, closedness and openness of branches and tableaux are defined like in Definition 7: A branch of a PUHR complement tableau is said to be *closed,* if it includes a node containing the atom $\bot$. A PUHR complement tableau is said to be closed if all its branches are closed. A branch (resp. PUHR complement tableau) which is not closed is said to be *open.*

Definition 21 *Let $\mathcal{S}$ be a set of range-restricted clauses in implication form and $\mathcal{A}$ a set of ground atoms, disjunctions, and clauses in implication form. $\mathcal{A}$ is said to be saturated with respect to $\mathcal{S}$ for the positive unit hyper-resolution and the complement splitting expansion rules when the following properties hold:*

- *if $(A_1 \wedge ... \wedge A_n \rightarrow E) \in \mathcal{S}$, $B_1 \in \mathcal{A}, ..., B_n \in \mathcal{A}$, and $(A_1 \wedge ... \wedge A_n)$ and $(B_1, ..., B_n)$ are unifiable, then $E\sigma \in \mathcal{A}$ for some most general unifier σ of $(A_1 \wedge ... \wedge A_n)$ and $(B_1, ..., B_n)$.*

- *If $(E_1 \vee E_2) \in \mathcal{A}$, then $\{E_1\} \cup Neg(E_2) \subseteq \mathcal{A}$, or $E_2 \in \mathcal{A}$.*

Note that if $\mathcal{A}$ is saturated with respect to $\mathcal{S}$ for the positive unit hyper-resolution and the complement splitting expansion rules, then it is also saturated for the positive unit hyper-resolution and the splitting expansion rules.

Model soundness for PUHR complement tableaux follows from Theorem 14.

Lemma 22 *Let $\mathcal{S}$ be a set of clauses and $A_1, ..., A_n (n \geq 1)$ be atoms.*

1. *If M is a minimal Herbrand model of $\mathcal{S}$ such that $M \not\models A_1 \wedge ... \wedge A_n$, then M is a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$.*

2. *If M is a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$, then M is also a minimal Herbrand model of $\mathcal{S}$.*

Proof: 1. Let $H(\mathcal{M})$ be a nonminimal model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$. There exists $\mathcal{M}_1 \subset \mathcal{M}$ such that $H(\mathcal{M}_1)$ is a model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$. Hence, $H(\mathcal{M})$ is not a minimal model of $\mathcal{S}$.

2. Assume that $H(\mathcal{M})$ is a Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$ which is not a minimal Herbrand model of $\mathcal{S}$. There is $\mathcal{M}_1 \subset \mathcal{M}$ such that $H(\mathcal{M})$ is a model of $\mathcal{S}$. Since $H(\mathcal{M}) \not\models A_i$ for some $i = 1, ..., n$ and since $\mathcal{M}_1 \subset \mathcal{M}$, $H(\mathcal{M}_1) \not\models A_i$. $H(\mathcal{M}_1)$ is therefore not a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$, and the same holds of $H(\mathcal{M})$. ■

Lemma 23 *Let $\mathcal{E}$ be a set of clauses in implication form, ground atoms and disjunctions of ground atoms, $E_1 \vee E_2 \in \mathcal{E}$ be a ground clause, and $\mathcal{M}$ be a set of ground atoms. $H(\mathcal{M})$ is a minimal model of $\mathcal{E}$ if and only if*

1. *either it is a minimal model of $\mathcal{E} \cup \{E_1\} \cup Neg(E_2)$*
2. *or it is a minimal model of $\mathcal{E} \cup \{E_2\}$ and for all $\mathcal{M}_1 \subseteq \mathcal{M}$, $H(\mathcal{M}_1)$ is not a minimal model of $\mathcal{E} \cup Neg(E_2)$.*

Proof: Let $H(\mathcal{M})$ be a minimal model of $\mathcal{E}$. If $H(\mathcal{M})$ does not satisfy E_2, then $H(\mathcal{M})$ is a model of $\mathcal{E} \cup \{E_2 \rightarrow \perp\}$. By Lemma 22, $H(\mathcal{M})$ is a minimal model of $\mathcal{E} \cup Neg(E_2)$. If $H(\mathcal{M})$ satisfies E_2 it is a model of $\mathcal{E} \cup \{E_2\}$. If it is not a minimal model of $\mathcal{E} \cup \{E_2\}$, then there exists $\mathcal{M}_1 \subset \mathcal{M}$ such that $H(\mathcal{M}_1)$ is a model of $\mathcal{E} \cup \{E_2\}$, hence of $\mathcal{E}$, contradicting the hypothesis that $H(\mathcal{M})$ is a minimal model of $\mathcal{E}$. By Lemma 22, if $H(\mathcal{M})$ is a minimal model of $\mathcal{E} \cup Neg(E_2)$, then it is also a minimal model of $\mathcal{E}$. Let $H(\mathcal{M})$ be a minimal model of $\mathcal{E} \cup \{E_2\}$. If $H(\mathcal{M})$ is not a minimal model of $\mathcal{E}$, then there exists $\mathcal{M}_1 \subset \mathcal{M}$ such that $H(\mathcal{M}_1)$ is a minimal model of $\mathcal{E}$. Since $H(\mathcal{M})$ is a minimal model of $\mathcal{E} \cup \{E_2\}$, $H(\mathcal{M}_1)$ does not satisfy E_2. Since $E_1 \vee E_2$ in $\mathcal{E}$, $H(\mathcal{M}_1)$ satisfies E_1 . Therefore, $H(\mathcal{M}_1)$ satisfies $\mathcal{E} \cup \{E_2 \rightarrow \perp\}$, i.e. there exists $\mathcal{M}_2 \subseteq \mathcal{M}_1 \subseteq \mathcal{M}$, such that $H(\mathcal{M}_2)$ is a minimal model of $\mathcal{E} \cup Neg(E_2)$. ■

Theorem 24 (Minimal model completeness for complement tableaux) *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form, T be a fair PUHR complement tableau for $\mathcal{S}$, and $\mathcal{M}$ a set of ground atoms. If $H(\mathcal{M})$ is a minimal model of $\mathcal{S}$, then there is a branch $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) = \mathcal{M}$.*

Proof: Follows from Corollary 16 since by definition every PUHR complement tableau for a set $\mathcal{S}$ can be constructed from a PUHR (noncomplement) tableaux by adding $\perp$ to some of its nodes, and from Lemma 23 which basically states that minimal models are preserved by complement splitting. ■

The following example shows that complement splitting is not always sufficient to prune all nonminimal models.

Example 5 Let $\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow P(b) \vee P(f(x)), P(a) \rightarrow P(b)\}$. Let T be the PUHR complement tableau for $\mathcal{S}$ by applying first the PUHR rule on $\top \rightarrow P(a)$ and $P(a) \rightarrow P(b)$, and then alternatively the PUHR and splitting rule on $P(x) \rightarrow P(b) \vee P(f(x))$. Although $H(\{P(a), P(b)\})$ is the unique minimal model of $\mathcal{S}$, T also has branches corresponding to the models $H(\{P(a), P(b), P(f(a)), ..., P(f^n(a))\})$ for all $n \in \mathbf{N}$.

Although possibly having branches corresponding to nonminimal models, PUHR complement tableaux never have two distinct branches defining the same model, as established next.

Lemma 25 *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form, T be a fair PUHR complement tableau for $\mathcal{S}$, and $\mathcal{B}_L$ and $\mathcal{B}_R$ be two open branches of T. If $\mathcal{B}_L$ appears to the left of $\mathcal{B}_R$ in T, then $Atoms(\cup\mathcal{B}_R) \not\subseteq Atoms(\cup\mathcal{B}_L)$.*

Proof: Let A_R be an atom in the first node of $\mathcal{B}_R$ (in a root to leaf traversal) which is not not in $\mathcal{B}_L$. By definition of the complement splitting rule, $(A_R \rightarrow \bot) \in \cup\mathcal{B}_L$. Hence $A_R \notin \cup\mathcal{B}_L$. ∎

Corollary 26 *Let $\mathcal{S}$ be a satisfiable set of range-restricted clauses in implication form, T be a fair PUHR complement tableau for $\mathcal{S}$ and $\mathcal{B}_0, ..., \mathcal{B}_i, ...$ a left-to-right enumeration of the open branches of T.*

1. *$H(Atoms(\cup\mathcal{B}_0))$ is a minimal model of $\mathcal{S}$.*
2. *If $i \neq j$, then $Atoms(\cup\mathcal{B}_i) \neq Atoms(\cup\mathcal{B}_j)$*

Proof: 1. Since $\mathcal{B}_0$ is the leftmost branch of T, by Lemma 25 $H(Atoms(\mathcal{B}_0))$ is a minimal model of $\mathcal{S}$.

2. Follows directly from Lemma 25. ∎

The following example demonstrates that a PUHR complement tableau can generate nonminimal models.

Example 6 Let $\mathcal{S}$ be the set of clauses of Example 3, i.e.:

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) & \qquad P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & \qquad P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Figure 4 gives a PUHR complement tableau for $\mathcal{S}$. The models generated by this PUHR complement tableau are $H(\{P(a), P(d)\})$, $H(\{P(b), P(c), P(a)\})$, $H(\{P(b), P(a)\})$, and $H(\{P(b), P(c), P(d)\})$. Note that although some are not minimal, the PUHR complement tableau returns no duplicates.

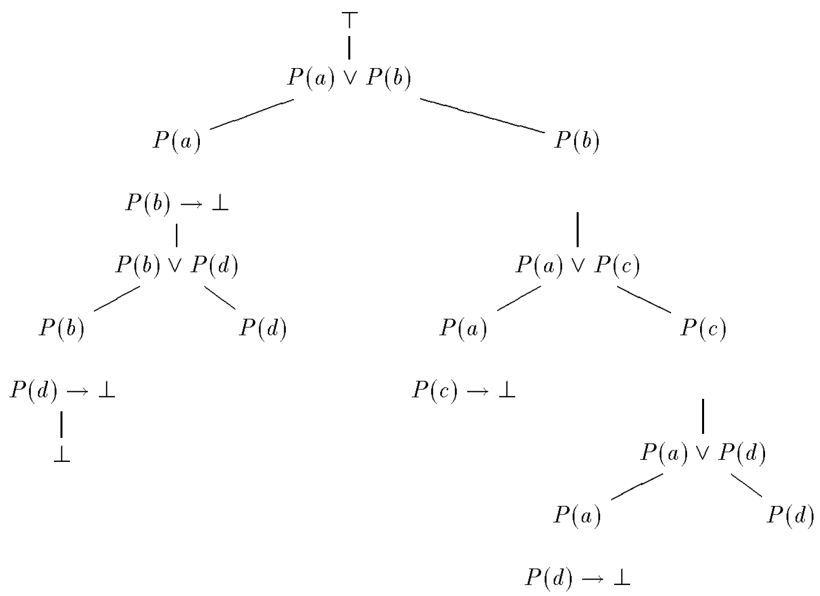


Figure 4: A PUHR complement tableau.

4.3 Implementation of Complement Splitting

Complement splitting can be built into SATCHMO by replacing the procedure `satisfy` by the following procedure `cs_satisfy`, as shown by Figure 5. `cs_component` returns not only the atoms of a disjunction, like `component` does, but also the rest of the disjunction on the right hand side of the returned atom (`false` if this right hand side is empty). This implementation, which we call CS-SATCHMO, departs slightly from Definition 19 since it represents $Neg(A_1 \vee ... \vee A_n)$ as $A_1 \vee ... \vee A_n \rightarrow \perp$ instead of $\{A_1 \rightarrow \perp, ..., A_n \rightarrow \perp\}$. Since the A_i are ground, the two representations are equivalent.

The complete program of CS-SATCHMO is given in Appendix A.

4.4 Constrained Search

By Corollary 26 the first model returned from a depth-first-left-first traversal of a PUHR complement tableau is minimal, and by Lemma 25 no models are $\leq$-larger than subsequently returned models. In order to prune PUHR complement tableaux from nonminimal models, it therefore suffices to constrain any model under construction not to be $\leq$-larger than any previously returned model. This is easily achieved by adding to the set of clauses a constraint $Neg(\{A_1, ..., A_n\}) = \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$ once a (finite) model $H(\{A_1, ..., A_n\})$ has been constructed.

```
cs_satisfy_all([]).
cs_satisfy_all([_B ---> H | Tail]) :-
     H,
     !,
     cs_satisfy_all(Tail).
cs_satisfy_all([_B ---> H | Tail]) :-
     cs_satisfy(H),
     cs_satisfy_all(Tail).

cs_satisfy(E) :-
     cs_component(Atom, Suffix, E),
     not (Atom = false),
     assume(Atom),
     assume_neg(Suffix).

cs_component(A, S, (A ; S)).
cs_component(A, S, (_ ; Rest)) :-
     !,
     cs_component(A, S, Rest).
cs_component(A, false, A).

assume_neg(false) :-
   !.
assume_neg(E) :-
   assume(E ---> false).
```

Figure 5: Complement splitting for SATCHMO

Definition 27 (Depth-first minimal model generation procedure) *Let $\mathcal{S}$ be a set of range restricted clauses in implication form. Applying the depth-first minimal model generation procedure to $\mathcal{S}$ consists in a depth-first-left-first construction of a fair PUHR complement tableau for $\mathcal{S}$ such that $\mathcal{S}$ is augmented with $Neg(\mathcal{M})$ after each computation of a model $H(\mathcal{M})$ of $\mathcal{S}$.*

Note that, by Definitions 7 and 19, if $\mathcal{S}_1$ and $\mathcal{S}_2$ are sets of range-restricted clauses in implication form such that $\mathcal{S}_1 \subseteq \mathcal{S}_2$ and all clauses in $\mathcal{S}_2 \setminus \mathcal{S}_1$ are of the form $A_1 \wedge ... \wedge A_n \rightarrow \perp$, then every PUHR complement tableau for $\mathcal{S}_2$ can be obtained from a PUHR complement tableau for $\mathcal{S}_1$ by adding $\perp$ to some nodes. Conversely, every PUHR complement tableau for $\mathcal{S}_1$ can be obtained from a PUHR complement tableau for $\mathcal{S}_2$ by discarding $\perp$ from some nodes.

Recall that a set of clauses is finitary if its minimal Herbrand models are all finitely representable.

Lemma 28 *Let $\mathcal{S}$ be a finitary and finite set of range-restricted clauses in implication form, and T be a PUHR complement tableau for $\mathcal{S}$.*

If t is a node in T, let $\mathcal{B}_0, ..., \mathcal{B}_{n_t}$ be branches of T to the left of t such that $H(Atoms(\cup\mathcal{B}_0)), ..., H(Atoms(\cup\mathcal{B}_{n_t}))$ are minimal models of $\mathcal{S}$.

Let T_t be the PUHR complement tableau for $\mathcal{S} \cup Neg(\cup\mathcal{B}_0) \cup ... \cup Neg(\cup\mathcal{B}_{n_t})$ corresponding to T. If $\mathcal{B}$ is a branch of T, let $\mathcal{B}_t$ denote the corresponding branch in T_t and conversely.

$\mathcal{B}_t$ is open in T_t if and only if $\mathcal{B}$ is open in T and $Atoms(\cup\mathcal{B}_i) \not\subseteq Atoms(\cup\mathcal{B}_t)$, for all $i = 0, ..., n_t$.

Proof: Assume that $\mathcal{B}$ is an open branch of T and $Atoms(\cup\mathcal{B}_i) \not\subseteq Atoms(\cup\mathcal{B})$, for all $i = 0, ..., n_t$. For all $i = 0, ..., n_t$ there exists an atom $A_i \in \cup\mathcal{B}$ such that $A_i \in \cup\mathcal{B} \setminus \cup\mathcal{B}_i$. Therefore, $H(Atoms(\cup\mathcal{B})) \models Neg(\cup\mathcal{B}_i)$. Hence $\mathcal{B}_t$ is open in T_t.

Assume that $\mathcal{B}_t$ is an open branch of T_t. If $Atoms(\cup\mathcal{B}_i) \not\subseteq Atoms(\cup\mathcal{B})$, for all $i = 0, ..., n_t$, then $\perp \notin \cup\mathcal{B}$. Hence $\mathcal{B}$ is open in T. ■

Theorem 29 (Soundness and completeness of the depth-first minimal model generation procedure) *Let $\mathcal{S}$ be a finite set of range-restricted clauses in implication form. If $\mathcal{S}$ is finitary, then applied on $\mathcal{S}$, the depth-first minimal model generation procedure terminates, returns all minimal models of $\mathcal{S}$ (i.e. it is complete), does not return any nonminimal model of $\mathcal{S}$ (i.e. it is sound), and does not return any minimal model more than once.*

Proof: Let $\mathcal{S}$ be a finitary and finite set of range restricted clauses in implication form.

Soundness: By Corollary 26 the first model returned by the procedure is a minimal model of $\mathcal{S}$. Assume that the first n models $H(\mathcal{M}_0), ..., H(\mathcal{M}_{n-1})$ returned by the procedure are minimal models of $\mathcal{S}$. Let T be the tableau

expanded so far. After returning the first n models, the procedure backtracks to a node t of T, such that the branches corresponding to previously returned models are to the left of t. The $(n+1)$-th model returned by the procedure corresponds to the first open branch of a tableau T_t for $\mathcal{S} \cup Neg(\mathcal{M}_0) \cup ... \cup Neg(\mathcal{M}_{n-1})$. By Lemma 28, this model is not $\leq$-larger than any previously returned model. By Corollary 26 it is a minimal model of $\mathcal{S} \cup Neg(\mathcal{M}_0) \cup ... \cup Neg(\mathcal{M}_{n-1})$. Hence, by Lemma 22 it is a minimal model of $\mathcal{S}$ as well. By induction, all models returned are minimal models of $\mathcal{S}$.
Completeness: For any two minimal models $H(\mathcal{M}_1)$ and $H(\mathcal{M}_2)$ of $\mathcal{S}$, $\mathcal{M}_1 \not\subseteq \mathcal{M}_2$ and $\mathcal{M}_2 \not\subseteq \mathcal{M}_1$. Therefore, $H(\mathcal{M}_1) \models Neg(\mathcal{M}_2)$ and $H(\mathcal{M}_2) \models Neg(\mathcal{M}_1)$. Consequently, no branches corresponding to a minimal model $H(\mathcal{M})$ of $\mathcal{S}$ with $\mathcal{M} \notin \{\mathcal{M}_0, ..., \mathcal{M}_n\}$ of a PUHR complement tableau for $\mathcal{S}$ can be closed in a tableau for $\mathcal{S} \cup Neg(\mathcal{M}_0) \cup ... \cup Neg(\mathcal{M}_n)$, for some minimal models $H(\mathcal{M}_0), ..., H(\mathcal{M}_n)$ of $\mathcal{S}$. Since the procedure returns only minimal models, the result follows. From Lemma 28, it follows that no models are generated more than once.
Termination: Since $\mathcal{S}$ is finitary, it has by Theorem 18 finitely many minimal models. Since the procedure returns all and only minimal models of $\mathcal{S}$, and since no minimal models are generated more than once, the procedure terminates. ∎

The following example shows how the depth-first minimal model generation procedure generates only minimal models and does not return duplicates.

Example 7 Figure 6 gives the search spaces of the depth-first minimal model generation procedure for the set of clauses of Examples 3 and 6, i.e.:

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) & \qquad P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & \qquad P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Note that all models returned by the procedure are minimal.

It is worth noting that fairness is necessary for the depth-first minimal model generation procedure, as the following counter-example shows.

Example 8 Let $\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow P(f(x)) \vee P(b), P(a) \rightarrow P(b)\}$. An unfair PUHR complement tableau for $\mathcal{S}$ with leftmost branch $\{P(a), P(f(a)), ..., P(f^n(a)), ...\}$ not containing $P(b)$ does not return the minimal model $H(\{P(a), P(b)\})$ and does not give rise to applying the constraint $P(a) \wedge P(b) \rightarrow \bot$ for pruning redundant branches.

4.5 MM-SATCHMO

Figure 7 gives a program, we call MM-SATCHMO, which implements the depth-first minimal model generation procedure. It builds upon the implementation of complement splitting described in Section 4.2. A slight

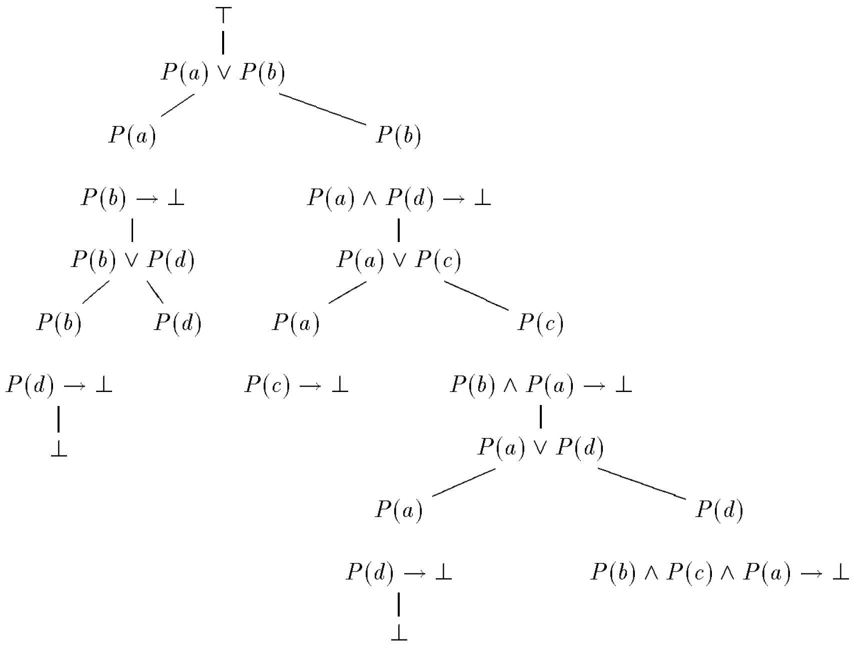


Figure 6: A run of the depth-first minimal model generation procedure.

modification of `satisfiable` suffices to construct the constraints induced by a (minimal) model.

The argument of the procedure `mm` is the body of the constraint under construction. This data structure is redundant, for the model under construction is also represented in the Prolog database. This redundancy can be easily removed, at the cost of a less readable program. A more serious source of inefficiency lies in the way how violated clauses are detected: the last inserted atoms are not used for an incremental detection. Although quite simple, an incremental evaluation requires longer and more complicated programs. An incremental clause evaluation turns out to be especially beneficial for the constrained search.

The complete program of MM-SATCHMO is given in Appendix B.

4.6 Breadth-First Minimal Model Generation

The depth-first minimal model generation procedure relies on chronological backtracking for introducing constraints ensuring the minimality of the subsequently generated models. If some minimal model $\mathcal{M}$ of the set $\mathcal{S}$ of clauses under consideration is infinite, then the depth-first minimal model generation procedure fails to generate those finite minimal models that were not constructed before $\mathcal{M}$. This can be avoided with a breadth-first expan-

```
minimal_model :-
     mm(true).

mm(C1) :-
    findall(Clause, violated_instance(Clause), Set),
    not (Set = []),
    !,
    mm_satisfy_all(Set, C1, C2),
    mm(C2).
mm(C) :-
    asserta(C ---> false).

mm_satisfy_all([], C, C).
mm_satisfy_all([_B ---> H | Tail], C1, C3) :-
     H,
     !,
     mm_satisfy_all(Tail, C1, C3).
mm_satisfy_all([_B ---> H | Tail], C1, C3) :-
     mm_satisfy(H, A),
     and_merge(A, C1, C2),
     mm_satisfy_all(Tail, C2, C3).

mm_satisfy(E, Atom) :-
     cs_component(Atom, Suffix, E),
     not (Atom = false),
     assume(Atom),
     assume_neg(Suffix).

and_merge(Atom, true, Atom) :-
     !.
and_merge(Atom, Conj, (Atom, Conj)).
```

Figure 7: The MM-SATCHMO program.

sion of PUHR tableaux.

The definitions of PUHR tableaux and PUHR complement tableaux in terms of two expansion rules, the PUHR rule and the splitting or complement splitting rule, was convenient so far, for it gives rise to rather simple proofs and appropriately conveys the intuition of the SATCHMO programs. In investigating the breadth-first expansion of PUHR tableaux or PUHR complement tableaux, it is convenient to rely on a sligthly more stringent definition of these tableaux based on a single expansion rule combining both, the positive unit hyper-resolution and the splitting or complement splitting rules.

Definition 30 (PUHR splitting and PUHR complement splitting rules) *Let $\mathcal{S}$ be a set of clauses in implication form.*

- *PUHR splitting rule:*

$$\begin{array}{c}
B_1 \\ \vdots \\ B_n \\ \hline
\begin{array}{c|c|c} E_1\sigma & \cdots & E_m\sigma \end{array}
\end{array}$$

- *PUHR complement splitting rule:*

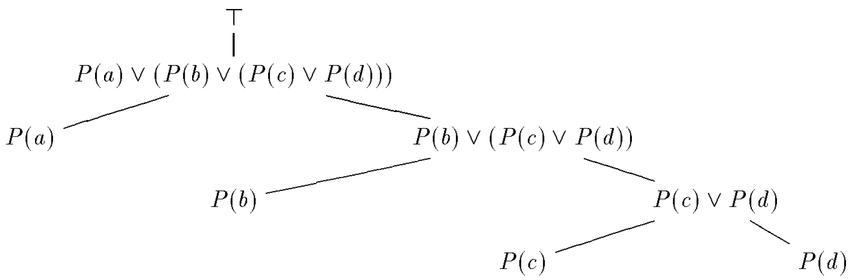
$$\begin{array}{c}
B_1 \\ \vdots \\ B_n \\ \hline
\begin{array}{l|c|l|c|r} E_1\sigma & \ldots & E_i\sigma & \ldots & E_m\sigma \\ Neg(E_2\sigma \vee \ldots \vee E_m\sigma) & & Neg(E_{i+1}\sigma \vee \ldots \vee E_m\sigma) & & \end{array}
\end{array}$$

In both rules, σ denotes a most general unifier of the body of a clause $(A_1 \wedge \ldots \wedge A_n \rightarrow E_1 \vee \ldots \vee E_i \vee \ldots \vee E_m) \in \mathcal{S}$ and of $(B_1, \ldots, B_n)$.

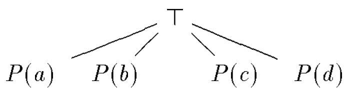
Definition 30 gives rise to revised definions of PUHR tableaux and of PUHR complement tableaux similar to Definition 7 and Definition 31:

Definition 31 (Revised PUHR (complement) tableaux) *PUHR (complement) tableaux for a set $\mathcal{S}$ of clauses in implication form are trees whose nodes are sets of ground atoms, disjunctions of ground atoms and ground implications of the form $A \rightarrow \bot$, resp. They are inductively defined as follows:*

1. *$\{\top\}$ is a revised PUHR (complement) tableau for $\mathcal{S}$.*



a. PUHR tableau for $\mathcal{S} = \{\top \rightarrow P(a) \vee (P(b) \vee (P(c) \vee P(d)))\}$.



b. Revised PUHR tableau for $\mathcal{S}$.

Figure 8: PUHR and revised PUHR tableaux compared.

2. *If T is a revised PUHR (complement) tableau for $\mathcal{S}$, if L is a leaf of T such that an application of the PUHR (complement) splitting rule to formulas in L yields m sets of formulas $S_1, ..., S_m$, then the tree T' obtained from T by adding the m nodes $L \cup S_1, ..., L \cup S_2$ as successors to L is a revised PUHR (complement) tableaux for $\mathcal{S}$.*

In contrast with the tableaux considered in the previous section, an atom is introduced at each node of a revised PUHR (complement) tableaux. This is illustrated by Figure 8. Revised PUHR (complement) tableaux are natural candidates for implementations. It is preferable to split a disjunction immediately after it has been introduced, indeed. Immediate splitting of an m-ary disjunction into m branches is also preferable to repeated splittings of binary disjunctions. In fact, the SATCHMO programs given so far do implement the PUHR splitting or PUHR complement splitting rules.

Theorem 32 *Under breadth-first expansion of a fair revised PUHR (complement) tableau:*

1. *The first model returned is minimal.*

2. *Let $\{H(\mathcal{M}_1), ..., H(\mathcal{M}_n)\}$ be the set of minimal models generated so far during a breadth-first expansion of a fair revised PUHR (complement) tableau. Any subsequently generated model $H(\mathcal{M})$ is minimal if and only if for all $i \in \{1, ..., n\}$, $\mathcal{M}_i \not\subset \mathcal{M}$.*

Proof: 1. A model returned is necessarily finite. Since an atom is introduced at each node of a revised PUHR (complement) tableau, the first model returned during a breadth-first expansion of a revised PUHR (complement) tableau necessarily has a minimal cardinality. It follows that it is minimal.
2. Let $\{M_1, ..., M_n\}$ be the set of minimal models generated so far during a breadth-first expansion of a fair revised PUHR (complement) tableau. Let $H(\mathcal{M})$ be the model returned next. $H(\mathcal{M})$ is a minimal model if for no (previously or subsequently) returned model $H(\mathcal{N})$, $N \subset M$. By hypothesis, this holds if $H(\mathcal{N})$ is a model returned by the procedure before $H(\mathcal{M})$, i.e. if $N = M_i$ for some $i \in \{1, ..., n\}$. Let $H(\mathcal{N})$ be a model returned by the procedure after $H(\mathcal{M})$. Since an atom is introduced at each node of a revised PUHR (complement) tableau and since the procedure expands the tableaux breadth-first, necessarily $|\mathcal{N}| \geq |\mathcal{M}|$. Hence, $\mathcal{N} \not\subset \mathcal{M}$. ∎

In the same spirit as with the depth-first minimal model generation procedure, and since the first model generated during a breadth-first expansion of revised PUHR (complement) tableaux is minimal, adding the same constraints as in the depth-first procedure prevents the generation of nonminimal as well as of duplicate minimal models without affecting the soundness and completeness properties of model generation. The result is a minimal model generation procedure capable of dealing with sets of clauses having infinite minimal models.

Definition 33 (Breadth-first minimal model generation procedure) *Let $\mathcal{S}$ be a set of range restricted clauses in implication form. Applying the breadth-first minimal model generation procedure to $\mathcal{S}$ consists in a breadth-first construction of a fair PUHR tableau or of a fair PUHR complement tableau for $\mathcal{S}$ such that $\mathcal{S}$ is augmented with $Neg(\mathcal{M})$ after each computation of a model $H(\mathcal{M})$ of $\mathcal{S}$.*

We want to emphasize that, in contrast to the depth-first minimal model generation procedure, the breadth-first minimal model generation procedure does not have to rely on complement splitting. However, relying on complement splitting in the breadth-first minimal model generation procedure guarantees that no duplicate models are produced, that the "leftmost model" is minimal and that no models can be subsumed by another "on its right". The last property indicates that even among the models generated so far we need to check against those "to the left" of the newly generated model. All this may result in substantial savings during the model computation process.

Additionally, since infinite models necessarily are "generated" last, we are guaranteed that the breadth-first minimal model generation procedure will eventually return all the finite minimal models of the considered set of clauses. A branch corresponding to a nonminimal infinite model $H(\mathcal{M}_\infty)$ is abandoned as soon as a finite minimal model $H(\mathcal{M})$ is produced such that

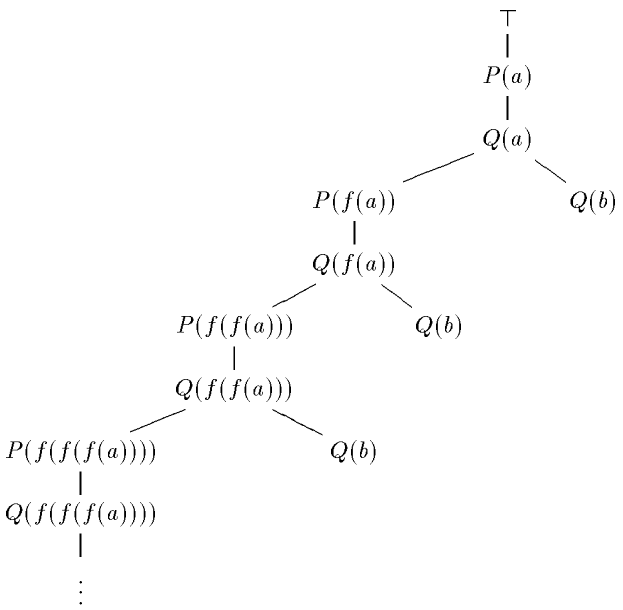


Figure 9: A revised PUHR tableau for the set of clauses of Example 9.

$\mathcal{M}$ is a subset of the already computed part of M_∞. Consider the following example:

Example 9 Let $\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow Q(x), P(x) \rightarrow P(f(x)) \vee Q(b)\}$.

$\mathcal{S}$ has an infinite minimal model, namely $H(\{P(a),\ Q(a),\ P(f(a)),\ Q(f(a)),\ P(f(f(a))),\ Q(f(f(a))), ...\})$ in addition to the finite minimal model $H(\{P(a),\ Q(a),\ Q(b)\})$. The revised PUHR tableau for $\mathcal{S}$ is given by Figure 9. Note that many models can be abandoned as a result of the constraint induced by the first minimal model $\{P(a), Q(a), Q(b)\}$ (No constraints are displayed in the figure). Applied on $\mathcal{S}$, the depth-first minimal model generation procedure is stuck on the infinite (minimal) model and does not return the finite minimal model.

5 Conclusions and Future Work

This paper presented two procedures for computing the minimal Herbrand models of sets of range restricted clauses. Both procedures are based on a positive unit hyper-resolution (PUHR) tableau method, which was introduced. The first minimal model generation procedure performs a depth-first expansion of PUHR tableaux relying on a form of backtracking involving constraints. The second minimal model generation procedure performs a breadth-first, constrained expansion of PUHR (complement) tableaux. Both procedures are optimal in the sense that each minimal model is constructed only once, and the construction of nonminimal models is interrupted as soon as possible. They are sound and complete in the following sense: The depth-first minimal model generation procedure computes all minimal Herbrand models of the considered clauses provided these models are all finite. The breadth-first minimal model generation procedure computes all finite minimal Herbrand models of the set of clauses under consideration. A compact implementation of the depth-first minimal model generation procedure in the form of a short Prolog program called MM-SATCHMO was also presented.

As a tableau procedure the proposed approach enjoys a good degree of efficiency stemming from its restricted search space, from limiting the applications of expansion rules and the use of matching without occur-check rather than full unification – see the performances reported in [29]. This is possible because, as a side-effect of a special range-restricted syntactical form, the generated tableaux are ground. Since it makes instantiation necessary, groundness of tableaux might be considered as a source of inefficiency in a refutation procedure. However, since Herbrand models are characterized as sets of ground atoms, this objection does not apply to a model generation procedure.

As model generation procedures, the approach proposed in this paper compares well with those reported in the literature, many of which are not sound in the sense that they generate nonminimal models [19, 12]. Compared with approaches based on model generation then testing for minimality [7, 20] the approach proposed here avoids nonminimal model generation altogether. The generation of nonminimal models is aborted as soon as possible, in general before they are fully developed. Also, the method we propose is applicable to first-order clauses and not confined to propositional or ground theories as the algorithms reported in [7, 35, 20]. While the applicability of the approach proposed in this article to sets of first-order clauses is a major advantage, most of the techniques increasing the efficiency for propositional or ground clauses proposed in [35, 20] can be incorporated into versions of the algorithms described here tailored for that case. Moreover, the approach proposed here requires no order to be placed on the sequence in which individual atoms are expanded – although such an order can be incorporated without substantial changes to the algorithm [35]. In [11] the concept of a *ghost tableau* is used to check the minimality of models that may be made nonminimal by the existential instantiation rule (or δ expansion [27]) in the (*primary*) tableau when testing for a "mini-consequence" property. The concept is useful when existential quantifiers are allowed in the theory which is not the case we consider in the present article.

Among the limitations of the procedures decribed here are their applicability only to range restricted and so called finitary sets of first-order clauses. However, range restriction is not much of a constraint, because a model preserving transformation of general clauses into range restricted ones was given. Moreover, most database and artificial intelligence applications naturally yield range-restricted specifications. We believe that much of real-life tasks enjoy the finiteness properties needed for the applicability of the depth-first minimal model generation procedure. For those applications with infinite minimal models, the breadth-first minimal model generation procedure can be applied for an exhaustive construction of all finite minimal models. One of the shortcomings of the procedures as reported here is their lack of incrementality. Further improvements, not discussed in this paper, can also be incorporated into the procedure. Another point is that, in some cases, the large number of constraints corresponding to generated minimal models may overwhelm the process without much positive contribution to discarding nonminimal models. A localized test that decides the minimality of the model based on the content of that model alone with no reference to other models can be found in [33]. Space considerations prevent us from detailing the approach here.

Testing of a prototype of the depth-first minimal model generation procedure points to its efficiency both as a model generator, and as a refutation system [29]. Indeed, the restriction to minimal models often dramatically reduces the search space, thus speeding up the closing of a tableaux. The prototype was able to deal with theories with a large number of minimal models with performances comparable to the best reported in the literature [20]. Further testing is needed to better evaluate the gains in performance and compare the minimal model generation procedure with existing systems. We plan also to further investigate applying a similar approach for query answering, integrity constraint enforcement, knowledge assimilation in data and knowledge base applications, as well as other possible approaches to testing model minimality.

Acknowledgments

We thank Norbert Eisinger, Heribert Schütz and Tim Geisler for the many fruitful discussions on the topic of this paper. Part of this research was done while the second author was visiting at Ludwig-Maximilians-Universität München on an Alexander von Humboldt Research Fellowship. The support of Alexander-von-Humboldt-Stiftung is appreciated.

References

[1] S. Abdennadher and H. Schütz. Model generation with existentially quantified variables and constraints. In *Proc. Sixth Int. Conf. on Algebraic and Logic Programming*, Springer-Verlag, LNCS, 1997.

[2] P. Baumgartner, U. Furbach, and I. Niemelä. A tableau calculus for diagnosis applications. In *Proc. Seventh Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Springer-Verlag, LNCS, 1997.

[3] F. Bry. Intensional updates: Abduction via deduction. In *Proc. Seventh Int. Conf. on Logic Programming,* MIT Press, 1990.

[4] F. Bry and A. Yahya. Minimal model generation with positive unit hyper-resolution tableaux. In *Proc. Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods,* Springer-Verlag, LNCS, 1996.

[5] M. Denecker and D. Schreye. A framework for indeterministic model generation with equality. In *Proc. Conf. on Fifth Generation Computer Systems*, 1992.

[6] R. Fagin, J.D. Ullman, and M.Y. Vardi. On the semantics of updates in databases. In *Proc. Second ACM Symp. on Principles of Database Systems*, 1983

[7] J.A. Fernández and J. Minker. Bottom-up evaluation of Hierarchical Disjunctive Deductive Databases. In *Proc. Eighth Int. Conf. on Logic Programming*, 660–675. MIT Press, 1991.

[8] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1987.

[9] P. Gardenförs. *Knowledge in Flux: Modeling the dynamic of epistemic states.* MIT Press, 1988.

[10] T. Geisler, S. Panne, and H. Schütz. Satchmo: The compiling and functional variants. *J. Automated Reasoning*, Vol. 18 No. 2, 227–236, 1997.

[11] J. Hintikka. Model minimization – an alternative to circumscription. *J. Automated Reasoning*, Vol. 4, 1–13, 1988.

[12] K. Inoue, M. Koshimura, and R. Hasegawa. Embedding negation as failure into a model generation theorem prover. In *Proc. Eleventh Int. Conf. on Automated Deduction*, 1992.

[13] R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *J. Automated Reasoning*, Vol. 13 No. 3, 297–338, 1994.

[14] J.W. Lloyd. *Foundations of logic programming.* Springer-Verlag, 1984, second edition 1987.

[15] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of disjunctive logic programming.* MIT Press, 1992.

[16] S. Lorenz. A tableau prover for domain minimization. *J. Automated Reasoning*, Vol. 13, 375–390, 1994.

[17] D. Loveland, D. Reed, and D. Wilson. SATCHMORE: SATCHMO with RElevancy. *J. Automated Reasoning*, Vol. 14, 325–351, 1995.

[18] R. Manthey and F. Bry. A hyperresolution-based proof procedure and its implementation in prolog. In *Proc. Eleventh German Workshop on Artificial Intelligence*, Springer-Verlag, LNCS, 456–459, 1987.

[19] R. Manthey and F. Bry. Satchmo: a theorem prover implemented in Prolog. In *Proc. Ninth Int. Conf. on Automated Deduction*, Springer-Verlag, LNCS, 456–459, 1988.

[20] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proc. Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Springer-Verlag, LNCS, 1996.

[21] N. Olivetti. Tableaux and sequent calculus for minimal entailment. *J. Automated Reasoning*, Vol. 9, 99–139, 1992.

[22] D. Poole, R. Aleliunas, and R. Goebel. THEORIST: A logical reasoning system for default and diagnosis. Technical Report, University of Waterloo, 1985.

[23] D. Prawitz. A new improved proof procedure *Theoria*, Vol. 26, 102–139, 1960.

[24] A. Ramsay. *Formal Methods in Artificial Intelligence.* Cambridge University Press, 1988, second edition 1989.

[25] R. Reiter. A theory of disgnosis from first principles. *Artificial Intelligence,* Vol. 32, 57–95, 1987.

[26] J.A. Robinson. Automatic deduction with hyper-resoultion. *Int. J. Computational Mathematics*, Vol. 1, 227–234, 1965.

[27] R. Smullyan. *First-Order Logic.* Springer-Verlag, 1968.

[28] M. Suchenek. First-order syntactic characterizations of minimal entailment, domain minimal entailment and herbrand entailment. *J. Automated Reasoning*, Vol. 10, 237–236, 1993.

[29] H. Schütz and T. Geisler. Efficient model generation through compilation. In *Proc. Thirteenth Conf. on Automated Deduction*, Springer-Verlag, LNCS, 433–447, 1996.

[30] M. Winslett. Reasoning about actions using a possible models approach. *Proc. Seventh Nat. Conf. on Artificial Inteligence*, 1988.

[31] G. Wrightson (Editor). Special issue on automated reasoning with analytic tableaux, Part I. *J. Automated Reasoning*, Vol. 13 No. 2, 173–281, 1994.

[32] G. Wrightson (Editor). Special issue on automated reasoning with analytic tableaux, Part II. *J. Automated Reasoning*, Vol. 13 No. 3, 283–421, 1994.

[33] A. Yahya. Model generation in disjunctive normal databases. Tech. Rep. PMS-96-10, Inst. für Informatik, Munich University, 1996. http:// www.informatik.uni-muenchen.de/ pms/ publikationen/ berichte/ PMS-FB-1996-10.ps.gz

[34] A. Yahya. Generalized Query Answering in Disjunctive Deductive Databases: Procedural and Nonmonotonic Aspects. In *Proc. Fourth Int. Conf. on Logic Programming and Nonmonotonic Reasoning*, Springer-Verlag, LNCS, 1997.

[35] A. Yahya, J.A. Fernandez, and J. Minker. Ordered model trees: A normal form for disjunctive deductive databases. *J. Automated Reasoning*, Vol. 13 No. 1, 117–144, 1994.

Appendix A: CS-SATCHMO

```
cs_satisfiable :-
     findall(Clause, violated_instance(Clause), Set),
     not (Set = []),
     !,
     cs_satisfy_all(Set),
     cs_satisfiable.
cs_satisfiable.

violated_instance(Body ---> Head) :-
     (Body ---> Head),
     Body,
     not Head.

cs_satisfy_all([]).
cs_satisfy_all([_B ---> H | Tail]) :-
     H,
     !,
     cs_satisfy_all(Tail).
cs_satisfy_all([_B ---> H | Tail]) :-
     cs_satisfy(H),
     cs_satisfy_all(Tail).

cs_satisfy(E) :-
     cs_component(Atom, Suffix, E),
     not (Atom = false),
     assume(Atom),
     assume_neg(Suffix).

cs_component(Atom, Suffix, (Atom ; Suffix)).
cs_component(Atom, Suffix, (_Atom ; Rest)) :-
     !,
     cs_component(Atom, Suffix, Rest).
cs_component(Atom, false, Atom).

assume(Atom) :-
     asserta(Atom).
assume(Atom) :-
     once(retract(Atom)),
     fail.

assume_neg(false) :-
     !.
assume_neg(E) :-
     assume(E ---> false).
```

Appendix B: MM-SATCHMO

```
minimal_model :-
     mm(true).


mm(C1) :-
     findall(Clause, violated_instance(Clause), Set),
     not (Set = []),
     !,
     mm_satisfy_all(Set, C1, C2),
     mm(C2).
mm(C) :-
     asserta(C ---> false).

violated_instance(Body ---> Head) :-
     (Body ---> Head),
     Body,
     not Head.

mm_satisfy_all([], C, C).
mm_satisfy_all([_B ---> H | Tail], C1, C3) :-
     H,
     !,
     mm_satisfy_all(Tail, C1, C3).
mm_satisfy_all([_B ---> H | Tail], C1, C3) :-
     mm_satisfy(H, A),
     and_merge(A, C1, C2),
     mm_satisfy_all(Tail, C2, C3).

mm_satisfy(E, Atom) :-
    cs_component(Atom, Suffix, E),
    not (Atom = false),
    assume(Atom),
    assume_neg(Suffix).

and_merge(Atom, true, Atom) :-
    !.
and_merge(Atom, Conj, (Atom, Conj)).

cs_component(Atom, Suffix, (Atom ; Suffix)).
cs_component(Atom, Suffix, (_Atom ; Rest)) :-
     !,
     cs_component(Atom, Suffix, Rest).
cs_component(Atom, false, Atom).

assume(Atom) :-
     asserta(Atom).
assume(Atom) :-
     once(retract(Atom)),
     fail.
```

```
assume_neg(false) :-
     !.
assume_neg(E) :-
     assume(E ---> false).
```