Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation

François Bry [1] Adnan Yahya [2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstraße 67, D – 80538 München, Germany
http://www.pms.informatik.uni-muenchen.de

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

Keywords: Minimal Model, Model Generation, Tableau, Proof Theory.

Abstract. Minimal Herbrand models of sets of first-order clauses 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 an approach for generating the minimal Herbrand models of sets of first-order clauses. The approach builds upon *positive unit hyperresolution (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 given and its performance on benchmark suites is reported. 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. The proposed procedures are compared with related work in terms of both principles and performance on benchmark problems.

1 Introduction

Generating Herbrand models of sets of first-order clauses 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 [40, 41], in some approaches to query answering [24, 36, 78, 77], for updating database facts and views [22, 28, 72, 6, 2], in artificial intelligence for solving design synthesis and diagnosis problems [54, 61, 53, 4], and in nonmonotonic reasoning [46, 34, 50, 49] – see also [60, 68]. Artificial intelligence production systems can be seen as minimal model generators for propositional or first-order logic Horn clauses.

The conventional tableaux methods [66, 25, 73, 74] are however inappropriate as model generation procedures because they often return redundant or nonminimal models [34, 50, 68, 42]. 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 point to the efficiency of the approach.

Both procedures are based on *positive unit hyperresolution tableaux* (short *PUHR tableaux*), a (novel) formalization of an approach first introduced with the theorem prover SATCHMO [44, 45]. 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 hyperresolution* and the *splitting* (a simple version of β expansion [66, 25]) rules, from *range-restricted* clauses. Range restrictedness is a syntactical property required in many applications, e.g. deductive database languages. A transformation of general clauses into range restricted clauses is described 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 hyperresolution makes it possible not to blindly instantiate universally quantified variables. Instead, it combines in one step instantiations (or γ expansions [66, 25]) and splittings (or β ex-

pansion [66, 25]), thus reducing the depth of PUHR tableaux. Thanks to range-restrictedness full unification is not needed for computing positive unit hyperresolvents. "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. The *complement splitting* rule (introduced under this name in [45], called "reduction" in [55] and "folding-down" in [39]) cuts out some branches leading to nonminimal models. Because PUHR tableaux are ground, complement splitting can be nicely and efficiently built 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. An interesting property is established: If all minimal Herbrand models of a set of clauses are finite, then they are finitely many. A variation, called MM-SATCHMO, of the SATCHMO program is given, which implements the depth-first minimal model generation procedure in Prolog. The previously mentioned property ensures the termination of this procedure, in case all minimal models are finite.

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 paper is as follows. Section 2 introduces terminology and notations, and defines range-restricted clauses. In Section 3, PUHR tableaux are introduced, they are compared with refutation methods, and their implementation in Prolog – the program SATCHMO – is recalled. Section 4 is devoted to model generation using PUHR tableaux. Soundness and completeness results are given and PUHR tableaux are compared with model generation methods. Section 5 defines the depth-first and breadth-first minimal model generation procedures as modified PUHR tableaux methods. In this section, finiteness properties are first investigated, complement splitting and its implementation are discussed, a minimal model generation procedure based on depth-first search is defined and its implementation in Prolog – the program MM-SATCHMO – is given, breadth-first minimal model generation is investigated, the proposed minimal model generation procedures are compared with related work, and finally the performance of MM-SATCHMO on benchmarks is reported. The last Section is a conclusion.

A preliminary version of this paper (whithout proofs and whithout Sections 3.2, 4.2, 5.6, 5.7, and 5.8) has been published in the Proceedings of the

Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods [12].

2 Preliminaries

2.1 Terminology and Notations

Throughout the paper usual terminology and notations are used, as in e.g. [66, 25]. 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 $\bot$. The logical connectives $\wedge$ and $\vee$ are assumed to be right associative, i.e. if $\theta = \wedge$ or $\theta = \vee$, then $L_1\theta L_2\theta \ldots \theta L_{n-1}\theta L_n$ denotes $(L_1\theta(L_2\theta \ldots \theta(L_{n-1}\theta L_n)\ldots))$.

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 repetitions in the sequence $(B_1, ..., B_n)$ are allowed, a conjunction $(A_1 \wedge .... \wedge A_n)$ might be unifiable with a sequence containing 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 variables occur 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 $B \in \mathcal{A}$. In this paper, a Herbrand interpretation $H(\mathcal{A})$ will be said to be *finitely representable* if $\mathcal{A}$ is finite. Note that "finite representability of Herbrand interpretations" can be found in the literature, e.g. in [23], with another meaning. 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 the following, a *tree* denotes a pair (V, E) such that V is a set – the elements of which are called *vertices* – and E is binary relation on V – the elements of which are called *edges* – containing no cycles and with respect to which V is connected. $Vertices(T)$ denotes the set of vertices and $Edges(T)$ the set of edges of a tree T. If T_1 and T_2 are trees, $T_1 \cup T_2$ is defined as (V, E) with $V = Vertices(T_1) \cup Vertices(T_2)$ and $E = Edges(T_1) \cup Edges(T_2)$.

In this paper tableaux methods and minimal model generation procedures for sets of first-order clauses 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 $\top$. Note that clauses considered in many applications of minimal model generation – e.g. database view updates [6, 2], database schema design [7], abductive reasoning [6, 16, 17], diagnosis [54, 61, 53] – are range restricted. Also, clauses obtained from many-sorted formulas through the standard representation in first-order logic [18] are range restricted.

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:

$$RR(C) := \begin{cases} C & \text{if } C \text{ is range restricted;} \\ D(x_1) \wedge ... \wedge D(x_k) \wedge A_1 \wedge ... \wedge A_n \rightarrow B_1 \vee ... \vee B_m & \text{otherwise,} \end{cases}$$

where $x_1, ..., x_k$ are the variables occurring in the B_is and in none of the A_js.

Let $\mathcal{S}$ be a set of $\mathcal{L}$-clauses. For a term t distinct from a variable and 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 $RR(\mathcal{S}) = \{P(x, y) \rightarrow P(f(x), y)\ ,\ \top \rightarrow D(a)\ ,\ D(x) \rightarrow D(f(x))\}$.

Example 1 shows that, if the range restriction transformation is applied to a set of clauses that are already range restricted, a set of range restricted clauses is obtained which is not identical with the initial set. Note that the properties of the PUHR tableaux method and of the minimal model generation procedures given below only require that the method and procedures are applied to sets $\mathcal{S}$ of range restricted clauses but not that $\mathcal{S} = RR(\mathcal{S})$.

The following theorem shows that the range restriction transformation preserves models and minimal Herbrand 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 a 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 (Herbrand model, minimal Herbrand model, resp.) 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 (Herbrand model, minimal Herbrand model, resp.) of $RR(\mathcal{S})$.

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

Proof: Point 1 follows immediately from Definition 2. For point 2 the nonemptiness of $\mathcal{S}'$ (cf. Definition 2) is necessary, because the clauses $RR(C)$ such that $RR(C) \neq C$ 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, however, 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 [66, 25] 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, i.e. as the 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 if, like considered in the present paper, Herbrand models are to be represented as sets of *ground* atoms.

Definition 4 (Positive unit hyperresolvent) *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 hyperresolvent of C and $B_1, ..., B_n$.*

Lemma 5 *The positive unit hyperresolvent 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 hyperresolvent of a range restricted clause in implication form and ground atoms. Indeed, half-way unification (or matching) suffices in computing a positive unit hyperresolvent of a range restricted clause in implication form and of ground atoms.

In the next section, positive unit hyperresolution tableaux are defined for range restricted clauses. This is not a significant restriction, for Definition 2 gives a transformation of (finite) sets of general clauses into (finite) sets of range-restricted clauses which preserves models and minimal Herbrand models in the sense of Theorem 3. Note that this transformation is *not* necessary for applying the model generation methods described below, if the considered clauses are already range restricted.

3 Positive Unit Hyperresolution Tableaux and SATCHMO

3.1 Positive Unit Hyperresolution Tableaux

Starting from the set $\{\top\}$, the PUHR tableaux method expands a tree – or positive unit hyperresolution (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 hyperresolution (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, thanks to the range restrictedness of clauses, the splitting rule is applied to *ground* disjunctions.

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

1. *$\{\top\}$ is a positive unit hyperresolution tableau for $\mathcal{S}$.*
2. *If T is a positive unit hyperresolution 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 hyperresolution tableau for $\mathcal{S}$.*

Infinite PUHR tableaux for $\mathcal{S}$ are defined as follows: If $(T_i)_{i\in\mathbb{N}}$ is an infinite sequence of finite PUHR tableaux for $\mathcal{S}$ such that for all $i \in \mathbb{N}$ T_{i+1} results from an application of a PUHR tableau expansion rule to T_i, then $T = \bigcup_{i\in\mathbb{N}} T_i$ – i.e. the tree T with $Vertices(T) = \bigcup_{i\in\mathbb{N}} Vertices(T_i)$ and $Edges(T) = \bigcup_{i\in\mathbb{N}} Edges(T_i)$ – is a PUHR tableau for $\mathcal{S}$.

A branch of a positive unit hyperresolution tableau is said to be closed, if it includes a node containing the atom $\bot$. A positive unit hyperresolution 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 hyperresolution 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.

If $\mathcal{P}$ is a branch or a path from the root to a node N, then $\cup\mathcal{P}$ will denote the union of the nodes in $\mathcal{P}$. Note that if $\mathcal{P}$ is a path from the root to a node N of a PUHR tableau, then by Definition 7, $N = \cup\mathcal{P}$.

Convention. If N_1 and N_2 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. This ordering induces an ordering on the branches of a PUHR tableau in the natural way. Note that this ordering – of nodes or branches of a PUHR tableau – is independent from any strategy under which the PUHR tableau can be built. Expressions such as "a node appearing to the left of another node in a PUHR tableau" (cf. Theorem 25) or "the leftmost branch of a PUHR tableau" (cf. Example 3, Corollary 26, and Example 7) will refer to this ordering, not to an ordering induced by a search strategy.

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

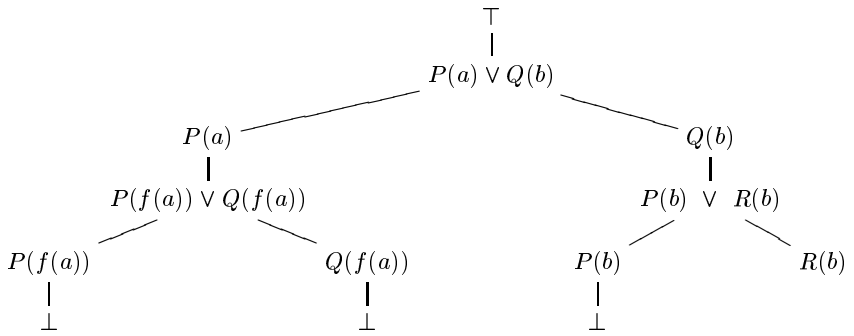


Figure 1: A PUHR tableau for the set of clauses of Example 2.

$$\top \rightarrow P(a) \vee Q(b) \qquad P(b) \rightarrow \bot$$
$$P(x) \rightarrow P(f(x)) \vee Q(f(x)) \qquad P(f(x)) \rightarrow \bot$$
$$Q(x) \rightarrow P(x) \vee R(x) \qquad P(x) \wedge Q(f(x)) \rightarrow \bot$$

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 and Lemma 5 the nodes of a PUHR tableau 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 $\bot$ only contribute to close branches. Since negative formulas do not explicitly occur in PUHR tableaux, closure is simply detected by the presence of $\bot$, which is simpler than checking for atomic closure [25].

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 hyperresolution 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 hyperresolvent 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 tableau is fair if along each of its open branches, each possible application of an expansion rule, which yields an atom or a disjunction of atoms not subsumed by previously generated atoms, 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 referred 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 $\bot$, 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 hyperresolution 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 e.g. in [43].

3.2 Comparison of PUHR Tableaux With Related Refutation Methods

The PUHR tableaux are a formalization of the principle of the SATCHMO programs, one of them is recalled in the next section. Other formalizations of the SATCHMO approach to theorem proving can be found in [16, 33, 17, 12, 3, 37, 11, 71]. A further more or less implicit formalization is subjacent to [43]. In [71, 11], EP Tableaux are proposed that generalize PUHR Tableaux to nonclausal formulas with "restricted quantification". PUHR and hyper tableaux [3, 37] are more in the "tableaux style" (cf. [66, 25, 73, 74]) than the formalizations [16, 33, 17]. PUHR tableaux are simpler than hyper tableaux [3] in which negative literals resolved away during hyperresolution yield closed branches. PUHR and hyper tableaux [3] are closely related to the positive tableaux of [30] that are defined for ground or propositional logic clauses.

In [3], a refutation method à la SATCHMO is described, that does not require clauses to be range-restricted. Variables occurring in more than one positive literal of a clause are instantiated using the γ rule of standard tableaux methods [66, 25]. Variables occurring in at most one positive literal do not have to be instantiated, since splitting disjunctions in which such unbound variables occur does not compromise refutation correctness. As pointed out in [3], this optimization is particularly interesting, because it applies to Horn clauses that frequently appear in applications. Note however, that this optimization is not applicable to model generation if, as assumed in the present paper, Herbrand models are to be represented by the *ground* atoms they satisfy.

In [3], it is proposed to achieve fairness by iterative deepening on the maximal depth of terms occurring in the generated clause instances. This seems more convenient than the (iterative deepening based) backtracking of free variable tableaux [25, 5]. However, it is debatable whether it is not preferable to achieve fairness by "level saturation" as described in [44, 45] and below in Section 3.3.

Note also the interesting optimization called "level cut" suggested in [3], which can be applied to most tableaux methods used for refutation. The "level cut" optimization consists in discarding branchings if one of the branching subtrees can be closed without using the branching assumption. This optimization is not applicable to model generation if, as it is assumed here, Herbrand models are to be represented by all the ground atoms they satisfy.

The data structure "model tree" described in [41] is related to PUHR tableaux as follows: The tree consisting of the (open) branches corresponding to minimal models of a PUHR tableau induces – by node relabeling and chain compacting – a model tree. However, model trees are defined only for ground clauses.

```
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 fair SATCHMO program.

In [44, 45], where SATCHMO was first presented, it is described in terms of positive unit hyperresolution and splitting and not as a tableaux method. This presentation has been retained by most authors referring to SATCHMO or extensions of it, e.g. [60, 26, 67, 43, 31, 38]. In fact, SATCHMO has been conceived as a tableaux method, as early publications [9, 8] on this project report. This is because enhancing a tableaux method with resolution was a new idea and because tableaux methods were considered inefficient that this view is not explicitly mentioned in [44, 45].

3.3 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 [25] 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.

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. $\bot$ 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 previously inserted into the Prolog database. For each ground instance `_B ---> H` of a clause returned by 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 fair SATCHMO, as well as variations of it have been first published in [44, 45]. In these articles, the programs are explained in more detail and performance on benchmark examples is reported.

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 the variable `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 the variable `Set` to the list:

```
[(true ---> p(b) ; 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.

4 Model Generation With PUHR Tableaux

4.1 Soundness and Completeness Results

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 exists an open branch $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) \subseteq \mathcal{M}$.*

Proof: Let $\mathbf{B}$ be the set of branches $\mathcal{B}$ of T such that $Atoms(\cup\mathcal{B}) \not\subseteq \mathcal{M}$. If $\mathbf{B}$ is empty, the result is established. Assume that $\mathbf{B} \neq \emptyset$. By the axiom of choice, for each $\mathcal{B} \in \mathbf{B}$ there exists $A_\mathcal{B} \in Atoms(\cup\mathcal{B}) \setminus \mathcal{M}$. Let $\mathcal{S}' = \mathcal{S} \cup \{A_\mathcal{B} \rightarrow \bot \ : \mathcal{B} \in \mathbf{B}\}$. By definition of $\mathcal{S}'$, since no $A_\mathcal{B}$ is in $\mathcal{M}$, $H(\mathcal{M})$ is also a model of $\mathcal{S}'$. Furthermore T can be extended into a positive unit hyperresolution tableau T' for $\mathcal{S}'$ by adding $\bot$ to the successor nodes of those nodes of T that contain some $A_\mathcal{B}$. Let $\mathcal{B}'$ denote such an extension of the branch $\mathcal{B}$ in T'. By construction, if $\mathcal{B} \in \mathbf{B}$, then $\mathcal{B}'$ is a closed branch of T'. By Theorem 10, since $H(\mathcal{M})$ is a model of $\mathcal{S}'$ and T' is positive unit hyperresolution tableau for $\mathcal{S}'$, T' has an open branch, say $\mathcal{B}_0$. Since $\mathcal{B}_0$ is open, it is no branch $\mathcal{B}'$ of T' extending a branch $\mathcal{B}$ of T such that $\mathcal{B} \in \mathbf{B}$. Since all clauses of $\mathcal{S}$, whose heads are $\bot$, are also in $\mathcal{S}'$, $\mathcal{B}_0$ is also an open branch of T. Since $\mathcal{B}_0 \notin \mathbf{B}$, by definition of $\mathbf{B}$, $Atoms(\cup\mathcal{B}_0) \subseteq \mathcal{M}$. ∎

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 hyperresolution tableau for $\mathcal{S}$, and $\mathcal{M}$ a set of ground atoms. If $H(\mathcal{M})$ is a*

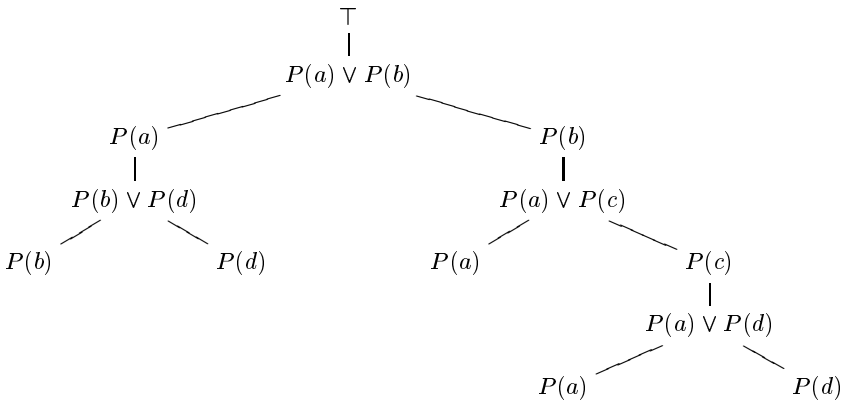


Figure 3: A PUHR tableau for Example 3 with nonminimal and duplicate models.

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:

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

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 [16, 17] and mentioned without proof in [36]. Since $\{\top\}$ is a PUHR tableau for every set $\mathcal{S}$ of clauses, fairness is clearly necessary in Corollary 16, although not in Theorem 15. A further interesting example is as follows.

Example 4 With the set of clauses

$$\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow P(f(x)) \vee P(b), P(a) \rightarrow P(b)\}$$

consistently expanding on the second clause will not allow the generation of the (only) minimal model $H(\{P(a), P(b)\})$ of $\mathcal{S}$.

4.2 Comparison With Other Model Generation Methods

As a model generation method, the PUHR tableaux method can be compared with model generators for given cardinalities. Possibly, one of the first such generator of models has been described in [35]. Nowadays, among the best known generators of finite models of (or up to) a given cardinality are FINDER [65] and SEM [79]. Their strength lies in a sophisticated very efficient implementation of the exhaustive search for models up to a given cardinality. The models generated by these methods are not necessarily minimal in the sense of the present paper. Moreover, they require to specify the cardinality of the universe. With PUHR tableaux, this is not necessary.

For ground clauses, the Davis-Putnam procedure [15] can be used as a model generator. A significant difference between PUHR tableaux and the trees expanded by the Davis-Putnam procedure is the PUHR rule which, also for ground clauses, gives a preference to positive atoms and expands the search space according to the implications. For applications such as e.g. query answering [24, 36, 78, 77], database fact and view updates [22, 28, 72, 6, 2], design synthesis and diagnosis [54, 61, 53, 4], this "positive preference" is a useful feature.

In [13, 70, 23, 52] tableaux methods are described that generate finite representations – in another sense than that considered in the present paper – for (possibly infinite) models. The method presented in [52] extracts models of possibly infinite tableaux branches by means of equational constraints. The methods [70, 23] make use of resolution and therefore are much more efficient than approaches based on the δ rule of classical tableaux methods. The method described in [70] applies only to the monadic Ackermann class. Like PUHR tableaux the method of [23] is based on positive hyperresolution but avoids splitting. In some cases this method builds finite representations of infinite models.

In [11, 71] an extension of the PUHR tableaux method is described which is complete for both, unsatisfiability and finite satisfiability. Completeness for finite satisfiability is achieved by generating models with minimal universes. This notion of "model minimality", which can be called "domain" or "universe minimality", is different and complementary to that investigated in the present article. For many applications – such as those addressed in [54, 61, 53, 6, 7, 20, 51, 2] – both notions of minimality, on the one hand domain minimality, on the other hand minimality of the set of satisfied ground atoms, are needed.

In [34, 42] a tableaux method is defined for first-order logic formulas which generates models with minimal universes by relying on so-called ghost subtableaux. Ghost subtableaux correspond to the extended $\exists$ or δ rule

of [11, 71]. Note, however, that the "universe minimality" of [34, 42] does not fully coincide with that of [11, 71]. In the implementation described in [42] the blind instantiation of the γ rule is controlled by giving a limit on the number of γ expansions for each γ formula.

Thus, the methods described in [34, 42, 11, 71] rely on an extended δ rule – also called δ^* rule – for processing existentially quantified variables. The approach investigated in the present paper in contrast relies on Skolemization.

Most forward chaining – also called bottom up – query answering methods for disjunctive databases, e.g. [57, 59, 77] can be seen as model generators similar to the PUHR tableaux methods. Like the PUHR tableaux method, these methods require the clauses to be range restricted and instantiate all variables. In [58, 32, 76], methods are proposed that, relying on forward chaining query answering methods for disjunctive databases, implement backwards chaining through an extension of the Magic Sets rewriting technique. These methods too can be seen as a tableaux method.

5 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 also discussed.

5.1 Finiteness Properties

Recall that a Herbrand interpretation $H(\mathcal{A})$ is called *finitely representable* if the set $\mathcal{A}$ of ground atoms it satisfies is finite.

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 $\mathbf{F}$ be the set of finitely representable minimal Herbrand models of $\mathcal{S}$. Assume $\mathbf{F}$ is infinite. If $\mathcal{A}$ is a finite set of atoms $\{A_1, ..., A_k\}$, let $Neg(\mathcal{A})$ denote the (singleton) set of clauses $\{A_1 \wedge ... \wedge A_k \rightarrow \bot\}$. For every finite subset F of $\mathbf{F}$, let $\mathcal{S}_F = \mathcal{S} \cup \bigcup\{Neg(\mathcal{A}) : H(\mathcal{A}) \in F\}$. By the axiom of choice, for every finite subset F of $\mathbf{F}$ there exists a minimal Herbrand model $H(\mathcal{M}_F)$ of $\mathcal{S}$ such that $H(\mathcal{M}_F) \in \mathbf{F} \setminus F$. Since all Herbrand models of $\mathcal{S}$ in $\mathbf{F}$ are minimal and since $H(\mathcal{M}_F) \notin F$, $H(\mathcal{M}_F)$ is a model of $Neg(\mathcal{A})$ for every $H(\mathcal{A}) \in F$. Therefore, $H(\mathcal{M}_F)$ is a model of $\mathcal{S}_F$. By the compactness theorem, $\mathcal{S}' = \bigcup\{\mathcal{S}_F : F \subset \mathbf{F}$ and F finite $\}$ is satisfiable. Since $\mathcal{S}'$ a set of clauses, it has a Herbrand model, and therefore also some minimal Herbrand model $H(\mathcal{M})$. By definition of $\mathcal{S}'$, $H(\mathcal{M}) \notin \mathbf{F}$. Therefore $\mathcal{M}$ is infinite. ∎

Conjectures. Although finite representability 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.

In applications, the finite representability of the minimal Herbrand models is often implicitly assumed. This is the case in particular of disjunctive databases [41] and of some forms of nonmonotonic reasoning [61, 34, 53, 60, 46, 49]. Thus, Theorem 18 is particularly interesting. Note that mentions of Theorems 17 and 18 or of similar results could not be found in the literature.

5.2 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 $Neg(C) := \{A_1 \rightarrow \bot, ..., A_n \rightarrow \bot\}$.

Definition 19 (Complement splitting rule)

$$\begin{array}{c|c} E_1 \vee E_2 & \\ \hline E_1 & E_2 \\ Neg(E_2) & \end{array}$$

The complement splitting rule is referred to under this name in [45]. It was inspired from the Davis-Putnam procedure [15] and from the "complement searching" technique of [56]. Other authors came to the same idea: Complement splitting is called "reduction" in [55] and "folding-down" in [39].

Like the splitting rule, the complement splitting rule is applied in the following definitions to *ground* disjunctions. Tableaux expanded with the positive unit hyperresolution 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 $A \to \bot$.

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

1. *$\{\top\}$ is a positive unit hyperresolution complement tableau for $\mathcal{S}$.*
2. *If T is a positive unit hyperresolution 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 hyperresolution complement tableau for $\mathcal{S}$.*

Infinite PUHR complement tableaux for $\mathcal{S}$ are defined as follows: If $(T_i)_{i \in \mathbb{N}}$ is an infinite sequence of finite PUHR complement tableaux for $\mathcal{S}$ such that for all $i \in \mathbb{N}$ T_{i+1} results from an application of the PUHR or complement splitting rule to T_i, then $T = \bigcup_{i \in \mathbb{N}} T_i$ is a PUHR complement tableau for $\mathcal{S}$.

Convention. The same convention is made for PUHR complement tableaux as for PUHR tableaux: If N_1 and N_2 are the nodes of a PUHR tableau T containing respectively $\{E_1, Neg(E_2)\}$ and $\{E_2\}$ and resulting from an application of the complement splitting rule to $E_1 \vee E_2$, the PUHR complement tableau T is ordered such that N_1 is the left sibling of E_2. This ordering

induces an ordering on the branches of a PUHR complement tableau which is independent from strategies under which the PUHR complement tableau can be built.

Note the following similarity between PUHR complement tableaux and the method proposed in [48]: For the leftmost open branch of a PUHR complement tableau, the condition expressed by complement splitting is equivalent to that expressed by the "groundedness test" of [48] (cf. also Section 5.7).

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 hyperresolution 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 hyperresolution and the complement splitting expansion rules, then it is also saturated for the positive unit hyperresolution 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 ground 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 $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$, then M is also a minimal Herbrand model of $\mathcal{S}$.*

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

2. Assume that ($\star$) $H(\mathcal{M})$ is a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$. If $H(\mathcal{M})$ is no minimal Herbrand model of $\mathcal{S}$ then there is $\mathcal{M}_1 \subset \mathcal{M}$ such that $H(\mathcal{M}_1)$ 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 a Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$. This contradicts the minimality of $H(\mathcal{M})$ assumed with ($\star$). ∎

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 $\mathcal{E} \cup \{E_2 \rightarrow \bot\}$. 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 $\mathcal{E} \cup \{E_2 \rightarrow \bot\}$, 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 $\mathcal{E} \cup Neg(E_2)$. ∎

For PUHR complement tableaux, fairness is defined similarly to fairness of PUHR tableaux: A PUHR complement tableau is said to be fair, if the union of the nodes of each of its open branches is saturated for the positive unit hyperresolution and complement splitting expansion rules.

Theorem 24 (Minimal model completeness of 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) tableau by adding $\bot$ to some of its nodes, and from Lemma 23 which basically states that minimal models are preserved by complement splitting. ∎

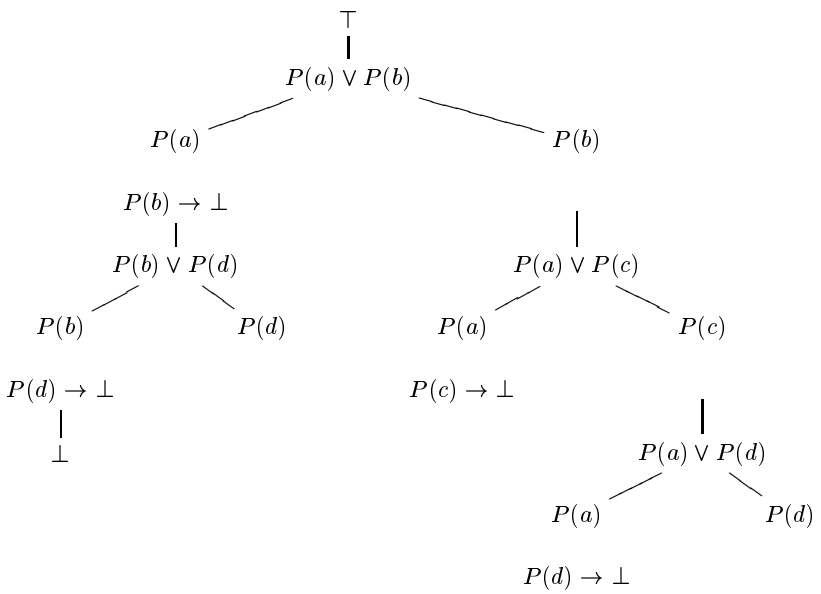


Figure 4: A PUHR complement tableau.

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

Example 5 Let $\mathcal{S}$ be the set of clauses of Example 3, i.e.:

$$\top \rightarrow P(a) \vee P(b) \qquad P(a) \rightarrow P(b) \vee P(d)$$
$$\top \rightarrow P(a) \vee P(c) \qquad P(b) \rightarrow P(a) \vee P(d)$$

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.

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 $A_R \notin \cup\mathcal{B}_L$. ∎

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

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

The procedures `assume` and `violated_instance` are defined like in fair SATCHMO (cf. Figure 2).

Figure 5: The CS-SATCHMO program

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. ∎

5.3 Implementation of Complement Splitting

Complement splitting can be built into SATCHMO by replacing the procedure `satisfy` by the `cs_satisfy` given in 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 $A_1 \vee ... \vee A_n \rightarrow \bot$ instead of $\{A_1 \rightarrow \bot, ..., A_n \rightarrow \bot\}$. Since the A_i are ground, the two representations are equivalent.

5.4 Constrained Depth-First Search for Minimal Model Generation

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 $Neg(\{A_1, ..., A_n\}) = \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$ once a (finite) model $H(\{A_1, ..., A_n\})$ has been constructed. In the following, such constraints are called "model constraints".

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 $\mathcal{S}$.*

As pointed out in Section 5.2, complement splitting has similarities with the "groundedness test" of [48]. This test can discard nonminimal models without relying on constraints $Neg(\mathcal{M})$ for each previously constructed minimal model $H(\mathcal{M})$. The price for this are on the one hand repeated computations of a test more complex than those performed by the depth-first minimal model generation procedure, on the other hand that repeated generations of the same minimal model are not precluded.

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 $A_1 \wedge ... \wedge A_n \rightarrow \bot$, then every PUHR complement tableau for $\mathcal{S}_2$ can be obtained from a PUHR complement tableau for $\mathcal{S}_1$ by adding $\bot$ 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 $\bot$ from some nodes.

Recall that a set of clauses is finitary if its minimal Herbrand models $H(\mathcal{M})$ are all finitely representable, i.e. such that $\mathcal{M}$ is finite.

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 $\bot \notin \cup\mathcal{B}$. 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 $\mathcal{S}$. From Theorem 24, it follows that the procedure returns all minimal models. From Lemma 28, it follows that no minimal 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 6 Figure 6 gives the search spaces of the depth-first minimal model generation procedure for the set of clauses of Examples 3 and 5, i.e.:

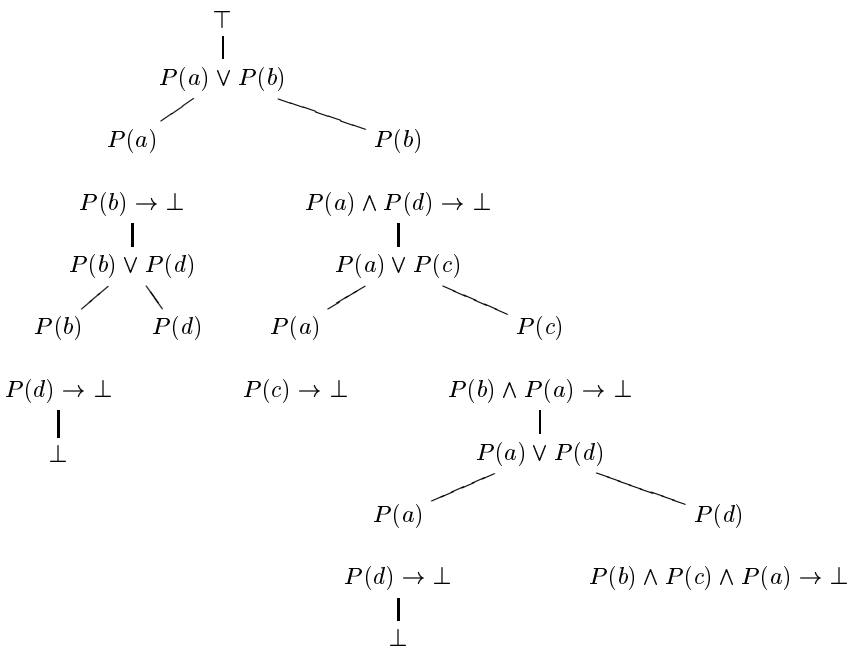


Figure 6: A run of the depth-first minimal model generation procedure.

$$\top \to P(a) \vee P(b) \qquad P(a) \to P(b) \vee P(d)$$
$$\top \to P(a) \vee P(c) \qquad P(b) \to P(a) \vee P(d)$$

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 7 Let $\mathcal{S} = \{\top \to P(a), P(x) \to P(f(x)) \vee P(b), P(a) \to P(b)\}$. 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 $P(a) \wedge P(b) \to \bot$ for pruning redundant branches.

5.5 MM-SATCHMO

Figure 7 gives the program MM-SATCHMO which implements the depth-first minimal model generation procedure. It builds upon the implementation of complement splitting described in Section 5.2. A slight modification of `satisfiable` suffices to construct the constraints induced by a (minimal) model.

```
minimal_model :- mm(true).

mm(_) :- false, !, fail.
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), assume(Atom),
     assume_neg(Suffix).

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

The procedures `assume` and `violated_instance` are defined like in SATCHMO (cf. Figure 2). The procedures `assume_neg`, and `cs_component` are defined like in CS-SATCHMO (cf. Figure 5).

Figure 7: The MM-SATCHMO program.

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.

5.6 Breadth-First Minimal Model Generation

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 $\mathcal{M}$. In this Section, it is shown how this can be avoided with a breadth-first expansion of PUHR tableaux. To this aim, revised definitions of PUHR tableaux and PUHR complement tableaux are convenient.

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

- *PUHR splitting rule:*

$$\frac{\begin{array}{c} B_1 \\ \vdots \\ B_n \end{array}}{E_1\sigma \mid \cdots \mid E_m\sigma}$$

- *PUHR complement splitting rule:*

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

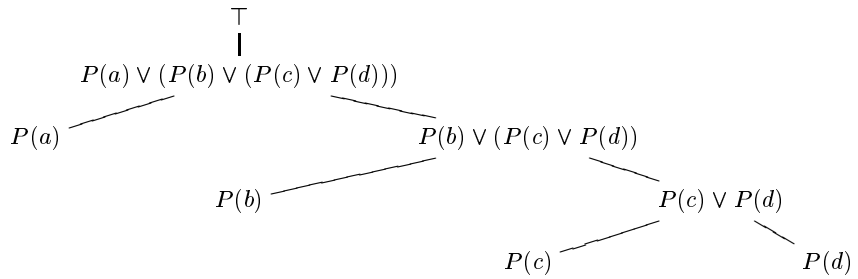
In both rules, σ denotes a most general unifier of the body of a clause $(A_1 \wedge ... \wedge A_n \rightarrow E_1 \vee ... \vee E_i \vee ... \vee E_m) \in \mathcal{S}$ and of $(B_1, ..., B_n)$.

Definition 30 gives rise to revised definitions of PUHR tableaux and of PUHR complement tableaux similar to Definition 7 and Definition 20:

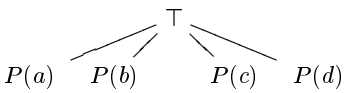
Definition 31 (Revised PUHR (complement) tableaux) *PUHR (complement) tableaux for a set $\mathcal{S}$ of clauses in implication form are (finite or infinite) trees whose nodes are sets of ground atoms, disjunctions of ground atoms and ground implications of the form $A \rightarrow \bot$, resp. Finite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

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

Infinite revised PUHR (complement) tableaux for $\mathcal{S}$ are defined as follows: If $(T_i)_{i \in \mathbb{N}}$ is an infinite sequence of finite revised PUHR (complement) tableaux for $\mathcal{S}$ such that for all $i \in \mathbb{N}$ T_{i+1} results from an application of the PUHR (complement) splitting rule to T_i, then $T = \bigcup_{i \in \mathbb{N}} T_i$ 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.

Convention. The same convention is made for revised PUHR (complement) tableaux as for (complement) PUHR tableaux: The immediate successors $N_1, N_2, \ldots, N_n$ of a node in a revised PUHR (complement) tableau T resulting from an application of the PUHR (complement) splitting rule are assumed to be ordered from left to right like the formulas they are defined from are ordered in the PUHR (complement) splitting rule of Definition 30.

Closedness and openness of branches or tableaux as well as fairness are defined for revised PUHR (complement) tableaux like for PUHR (complement) tableaux (cf. Definitions 7 and 11).

In contrast with the tableaux considered in the previous sections, an atom is introduced at each node of a revised PUHR (complement) tableau. This is illustrated by Figure 8.

Theorem 32 *Under breadth-first expansion of a fair revised PUHR (complement) tableau for a set $\mathcal{S}$ of clauses:*

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. Every model returned is necessarily finite. Since an atom is introduced at each node of a revised PUHR (complement) tableau, the first

Herbrand model $H(\mathcal{M})$ returned during a breadth-first expansion of a revised PUHR (complement) tableau T for $\mathcal{S}$ necessarily corresponds to an open branch $\mathcal{B}$ of T with minimal length. Therefore, there are no Herbrand models $H(\mathcal{N})$ of $\mathcal{S}$ such that $\mathcal{N} \subset \mathcal{M}$, i.e. $H(\mathcal{M})$ 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 tableaux breadth-first, necessarily $|\mathcal{N}| \geq |\mathcal{M}|$. Hence, $\mathcal{N} \not\subset \mathcal{M}$. ∎

Note that while in the previous sections the formalization of PUHR tableaux in terms of two expansion rules gives rise to a simpler treatment, the formalization in terms of revised PUHR tableaux is much more convenient for Point 1 of Theorem 32.

Since the first model generated during a breadth-first expansion of a revised PUHR (complement) tableau is minimal, adding the same "model 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 revised PUHR tableau or of a fair revised 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, 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".

Since infinite models necessarily are "generated" last, 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 $\mathcal{M}$ is a subset of the already computed part of M_∞, as the following example illustrates.

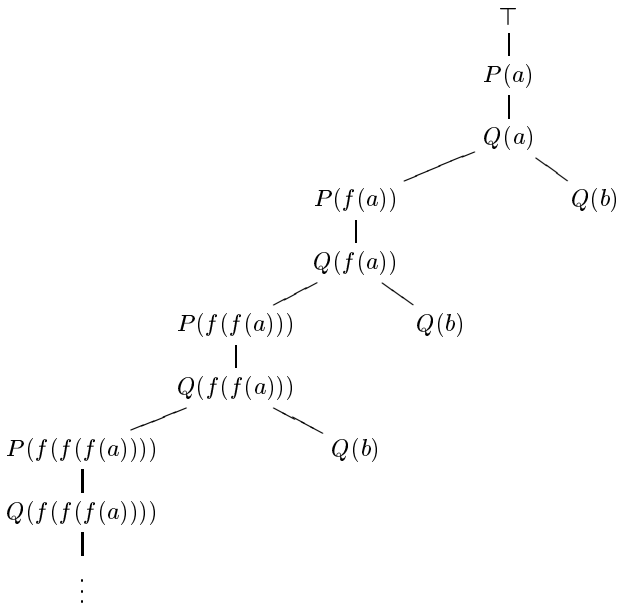


Figure 9: A revised PUHR tableau for the set of clauses of Example 8.

Example 8 Let $\mathcal{S} = \{\top \rightarrow P(a), P(x) \rightarrow Q(x), P(x) \rightarrow P(f(x)) \vee Q(b)\}$. $H(\{P(a), Q(a), P(f(a)), Q(f(a)),\ P(f(f(a))), Q(f(f(a))), ...\})$ is an infinite minimal model of $\mathcal{S}$ and $H(\{P(a),\ Q(a),\ Q(b)\})$ is a finite minimal model of $\mathcal{S}$. The revised PUHR tableau for $\mathcal{S}$ is given by Figure 9 (no constraints are displayed in the figure). Note that many models can be abandoned as a result of the constraint induced by the first minimal model $\{P(a), Q(a), Q(b)\}$ 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.

5.7 Comparison With Other Minimal Model Generators

In [48], a minimal model generation method for propositional logic is proposed. Like the approaches described here, the approach of [48] is a tableaux method. Unlike the the approaches described here, it performs no direct comparisons of minimal models specified by different branches. Instead, it relies on a "groundedness test" based on the following property. A Herbrand model $H(\mathcal{M})$ of a set $\mathcal{S}$ of ground clauses is minimal if and only if

$$(\star\star) \quad \forall A \in \mathcal{M} \quad \mathcal{S} \cup \overline{\mathcal{M}} \models A$$

where $\overline{\mathcal{M}}$ denotes $\{\neg B \mid B \text{ ground atom and } B \notin \mathcal{M}\}$. As already pointed out in Section 5.2, for the leftmost open branch of a PUHR complement

tableau, the condition expressed by complement splitting is equivalent to the "groundedness test" ($\star\star$). Note that while complement splitting is syntactically defined, the "groundedness test" is a model theoretic condition. This might make it less immediate to check than complement splitting. As opposed to the combination of complement splitting and "model constraints" proposed here, the "groundedness test" does not preclude repeated generations of the same minimal model. Because it relies on "model constraints", i.e. a form of memoization, the depth-first minimal model generation has, according to [48], an "exponential worst-case space complexity". In contrast, the method described in [48] is said there to have a "polynomial space complexity". A comparison of the run times of MM-SATCHMO with those reported in [48] is given below in Section 5.8: Both minimal model generation procedures achieve a comparable efficiency. It is a debatable question, which of the two approaches is preferable in practice. As it is often the case, the trade-off is between time and space: The one method saves computation time by storing results of previous computations, the other method relies on additional computations for avoiding any storage. For some applications, a method with restricted storage is needed. For others, storing minimal models might be preferable, e.g. if the minimal models have to be compared or further processed. Comparisons of minimal models are needed e.g. for comparing semantics of logic programs and deductive databases [40, 41], for comparing semantics of nonmonotonic reasoning [46, 68], for comparing answers to queries [24, 36, 78, 77], for choosing database (fact or view) updates [22, 28, 72, 6, 2], and for comparing alternative solutions to design and diagnosis problems [54, 61, 53, 4].

In [50], "minimal entailment" for propositional logic is investigated. A formula B is "minimaly entailed" by a formula A, if no minimal models of A falsify B. It is proposed in [50] to establish this property using two special tableaux methods. The first one, the "Algorithm TABLEAU for $A \vdash^T B$" [50, p. 107], is a tableaux method for signed, free syntax propositional logic formulas. Nonminimal models are detected at step 4 of the algorithm, i.e. after each expansion of a branch, by a comparison of the atoms in this branch with the previously generated models. A further test, at step 6 of the algorithm, is necessary for discarding so-called ignorable branches containing meaningless combinations of signed literals. The second tableaux method proposed with Definition 9 [50, p. 110] is an improvement of the previously mentioned algorithm for those cases where A is a set of (propositional logic) clauses, and B is a single (propositional logic) clause. The improvement basically consists in simpler expansion rules for the restricted syntax and, more importantly, in the addition of positive unit resolution (through Rule **R3** [50, p. 110]). Referring to the first method, the author of [50] writes: "we regard it mainly as a theoretical tool". Techniques such as the "groundedness test" of [48] or complement splitting that speed up the abandonment of branches corresponding to nonminimal or redundant

models are not considered in [50]. Moreover, it is questionable whether considering signed formulas does not introduce an overhead compared with tableaux methods for unsigned formulas [66, 25, 73, 74, 27].

Some deductive database query answering methods can be used for generating minimal models. The system DisLog [63, 64] implements several query answering methods for disjunctive databases [41]. Its forward chaining procedure can be used as a minimal model generator similar to the breadth-first minimal model generation of Section 5.6, although without special treatment of negative clauses, i.e. clauses all literals of which are negative. Moreover, DisLog proceeds by first generating (possibly nonminimal) models, then test for minimality. It therefore explores in general more interpretations than the approaches presented here and in [50, 21, 48]. The approach of [78] to constructing so-called "ordered minimal model trees" can as well be applied to generate minimal Herbrand models. This approach is however restricted to ground disjunctive deductive databases. This restriction makes it possible to simplify the considered clauses at every assignment of a truth value to an atom. It also demands that a fixed order, albeit not necessarily known in advance, for atom expansion be defined to achieve the uniqueness of the constructed tree under the given ordering.

Most semantics proposed for nonmonotonic reasoning – cf. e.g. [46, 68] – rely more or less explicitly on notions of model minimality. Thus, methods like e.g. [14, 47] for computing models according to such semantics can, with more or less adaptations, be applied to computing minimal Herbrand models in the sense of this paper. However, most such methods do not fully address the issues investigated here. Indeed, as explained e.g. in [68, Section 5, p. 251], they have to cope with notions such as "default" or "negation as failure" that are not relevant to the generation of minimal Herbrand models of sets of first-order clauses. Many of them, like e.g. the method described in [14], are only applicable to normal logic programs, i.e. they cannot cope with non-Horn clauses. The method of [14] is in addition restricted to ground clauses and makes use of this restriction like [78] for simplifying the considered clauses at every assignment of a truth value to an atom. Minimal model generators can be adapted to computing semantics for nonmonotonic reasoning, as shown e.g. in [49]. Note that most investigations of nonmonotonic reasoning, such as [68], are proof-theoretic in nature and neither rely on, nor specify algorithms for the generation of minimal models. In this respect, the article [50] is an exception: Although it is devoted to minimal entailment, it defines, as already mentioned, minimal model generation algorithms for propositional logic.

In [34, 42] an approach to "model minimization" is investigated. In fact, both articles [34, 42] are devoted to generating models with minimal universes, not to generating minimal Herbrand models in the sense considered here. The issue of "universe" or "domain minimization", also investigated in [10, 33, 68, 11, 71], is interesting, for two reasons. On the one hand, meth-

ods for universe minimization give rise to algorithms that are complete for both, unsatisfiability and finite satisfiability [11, 71]. On the other hand, the issue has practical applications, e.g. to designing database schemas [9, 7]. In [34, Section 9, p. 11] a modification of the tableaux method proposed there is sketched, so as to "minimize predicate extensions", i.e. to generate minimal Herbrand models in the sense considered in the present paper. This modification, which does not seem to be fully worked out, is basically in the spirit of complement splitting and of the constrained search as well as of the "groundedness test" of [48].

5.8 Experiments with MM-SATCHMO

In this Section, the performance of MM-SATCHMO on four benchmark suites, called A, B, D and F, are reported. Each suite includes 24 examples, each example consists of 5 to more than 100 000 clauses, each clause has up to 10 literals. The number of minimal models of an example ranges from 1 to 100 000.

The run times reported below have been obtained with MM-SATCHMO run under ECLiPSe Prolog Version 3.5.1 [19] on a Hewlett Packard Unix (HP-UX 10.20) Workstation HP Visualise C 160 (PA-8000 processor at 160 MHz, 192 MB RAM). Note that ECLiPSe Version 3.5.1 uses 32 bit words instead of 64 bit words as possible on a HP Visualise C 160.

ECLiPSe was started anew for each problem, thus avoiding any speed up or overhead resulting from a previously constructed symbol table or uncollected garbage. The reported CPU times were obtained using the `time` command of ECLiPSe.

The programs and benchmark suites referred to in this section are available at:

http://www.pms.informatik.uni-muenchen.de/software/MM-SATCHMO/

Worst-case Examples: The A Benchmark Suite. For nonnegative integers n and m, $A(n,m)$ denotes the set of n clauses of length m defined by:

$$A(n,m) := \{\ \texttt{true ---> a_i_1 ; ... ; a_i_m} \mid \texttt{i} = 1,\ldots,n\ \ \}$$

Applied on $A(n,m)$, SATCHMO computes m^n models by selecting an atom `a_i_j` for each `i` $\in \{1,\ldots,n\}$. Since `a_i_j` $\neq$ `a_h_k` for (`i`, `j`) $\neq$ (`h`, `k`), all models returned by SATCHMO are pairwise distinct and each of them is a minimal model of $A(n,m)$. Thus, for these examples, the additions MM-SATCHMO makes to SATCHMO have no effects. Therefore, examples of the A suite can be seen as worst-case examples for MM-SATCHMO.

The run times of MM-SATCHMO on the examples of the A suite are given by Table 1.

n \ m	3	4	5	6	7	8	9	10
3	0.01	0.02	0.05	0.11	0.22	0.40	0.72	1.22
4	0.03	0.15	0.60	2.08	7.48	22.57	58.34	134.76
5	0.15	1.44	13.74	85.05	413.62	1 445.91	4 782.78	13 317.30

Table 1: CPU times in seconds for computing all minimal models of $A(n, m)$.

Already for small n and m, computing all minimal models of $A(n, m)$ involves a tremendous potential search space. For computing the $5^4 = 625$ minimal models of $A(4, 5)$, truth value assignments for $4 \times 5 = 20$ propositional variables, i.e. $2^{20} = 1\ 048\ 576$ assignments, are possible. For computing the 10^5 minimal models of $A(5, 10)$, truth value assignments for 50 propositional variables, i.e. $2^{50} = 1\ 125\ 899\ 906\ 842\ 624$ (more than 1 million billions) assignments are possible. Of course, the search space actually expanded by MM-SATCHMO is significantly smaller: As soon as MM-SATCHMO assigns the value "true" to an `a_i_j`, it implicitly assigns the value "false" to all `a_i_k` such that `k` $\in \{1, \ldots, m\} \setminus \{$ `j` $\}$.

More informative than the overall time needed for computing all minimal models is the average time per minimal model. For example, if generating all minimal models of $A(5, 10)$ takes as much as 3 hours 42 minutes, each of the 100 000 minimal models of this example is computed on average in less than one and a half tenth of a second.

n \ m	3	4	5	6	7	8	9	10
3	0.37	0.31	0.40	0.51	0.64	0.78	0.99	1.22
4	0.37	0.58	0.96	1.60	3.12	5.51	8.90	13.48
5	0.62	1.41	4.40	10.94	24.61	44.12	81.00	133.17

Table 2: Average CPU times in 10^{-3} seconds for computing one minimal model of $A(n, m)$.

The (n, m) entry $t_2(n, m)$ of Table 2 is defined by $t_2(n, m) = \frac{t_1(n,m)}{m^n} \times 10^3$ where $t_1(n, m)$ denotes the (n, m) entry of Table 1.

Tables 1 and 2 suggest that $t_1(n, m) = O(m^{2n})$ and $t_2(n, m) = O(m^n)$. This can be confirmed as follows. In order to avoid a repeated generation of already returned models, MM-SATCHMO relies on adding "constraints", i.e. clauses with `false` as head, during complement splitting and after a minimal model is generated. We remind of the name of "model constraints" – cf. Section 5.4 – for those constraints introduced after the generation of minimal models . During the generation of all minimal models of $A(n, m)$

$$c_A(n, m) = \sum_{k=0}^{m^n-1} k = \frac{(m^n - 1) \times m^n}{2} = O(m^{2n})$$

evaluations of "model constraints" take place since no such constraints are present when the first minimal model is returned, and exactly $k-1$ such constraints are present when the k-th minimal model is generated. Thus, it is reasonable to assume that $t_1(n,m) = O(m^{2n})$. It follows that $t_2(n,m) = O(m^n)$ since, by definition, $t_2(n,m) = \frac{t_1(n,m)}{m^n} \times 10^3$.

Note that, since complement splitting introduces further constraints, more than 100 000 constraints are involved in the generation of all models of $A(5,10)$.

The Price of Constraints: The B Benchmark Suite. The large number of constraints is a source of inefficiency, because at each cycle of the main procedure of MM-SATCHMO, all constraints are evaluated. In order to estimate the cost of this evaluation, the B benchmark suite is now considered.

For nonnegative integers n and m, $B(n,m)$ denotes the set of clauses $A(n,m)$ augmented with the $m^n - 1$ model constraints that exclude all minimal models of $A(n,m)$ except the last one returned by MM-SATCHMO:

$$B(n,m) := C(n,m,m^n) \cup A(n,m)$$

where

$$C(n,m,k) := \{\ \texttt{false :- Mj}\ \mid\ \texttt{j} = 1,\ldots,k-1\ \}$$

and `Mj` denotes the conjunction of the atoms representing the `j`-th minimal model of $A(n,m)$ returned by MM-SATCHMO. Clearly, $B(n,m)$ has exactly one minimal model. Note that $B(n,m)$ consists of $n + m^n - 1$ clauses, e.g. $B(4,5)$ consists of 628, $B(5,10)$ of 100 004 clauses.

Following a basic optimization mentioned in [45], a model constraint is expressed as a Prolog clause `false :- Mj` instead of `Mj ---> false`. If `false` is derivable, this optimization avoids asserting `false` in the Prolog database just before retracting it while backtracking.

By definition of $B(n,m)$, generating the one minimal model of $B(n,m)$ with MM-SATCHMO amounts to generating all minimal model of $A(n,m)$ with MM-SATCHMO. While the constraints are progressively introduced during the generation of all models of $A(n,m)$, they are present from the beginning during the construction of the first (and only) model of $B(n,m)$. Comparing Table 1 with Table 3 below shows how this presence affects the run times.

m / n	3	4	5	6	7	8	9	10
3	0.01	0.01	0.03	0.08	0.16	0.32	0.60	1.06
4	0.02	0.10	0.49	2.07	7.95	24.81	66.35	162.20
5	0.10	1.29	14.93	102.43	451.75	1 799.27	5 737.51	15 398.20

Table 3: CPU times in seconds for computing the first (and only) minimal model of $B(n,m)$.

In order to estimate the overhead introduced by the model constraints, one has to consider the number of times such constraints are evaluated. As already observed, for $A(n,m)$ this number is

$$c_A(n,m) = \sum_{k=0}^{m^n-1} k$$

For $B(n,m)$, it is

$$c_B(n,m) = \left(\sum_{k=0}^{m^n-1} k\right) + m^n - 1 = c_A(n,m) + m^n - 1$$

for the following reasons. The set $C(n,m,m^n)$ of model constraints of $B(n,m)$ was generated by running MM-SATCHMO on $A(n,m)$ and the model constraints were stored using `asserta` in the order of their generation. Thus, a Prolog call to `false` evaluates the model constraints in the reverse order of their generation. While computing the single model of $B(n,m)$, all the m^n-1 model constraints in $B(n,m)$ are evaluated after the first interpretation is generated. The bodies of all these constraints but the last one evaluate to "false". After the k-th ($2 \leq k \leq m^n-1$) interpretation is constructed, only m^n-k model constraints need to be evaluated, because the body of the (m^n-k)-th model constraint evaluates to "true", thus deriving the atom `false`. When the (m^n)-th interpretation, i.e. the single model of $B(n,m)$, is generated, all the m^n-1 model constraints in $B(n,m)$ are evaluated once again, the body of all of them evaluating to "false".

Table 4 gives, for those values of n and m for which the estimations make sense, the overall times spent for one evaluation of the clauses in $C(n,m,m^n)$ during the computation of the single model of $B(n,m)$, i.e. the times spent for one evaluation of all model constraints generated during the computation of all models of $A(n,m)$.

m / n	5	6	7	8	9	10
4	-	-	0.48	2.25	8.00	27.40
5	1.19	17.42	38.15	353.23	954.81	2 080.10

Table 4: CPU times in seconds spent for one evaluation of all clauses in $C(n,m,m^n)$ during the computation of the first (and only) minimal model of $B(n,m)$.

The (n,m) entry $t_4(n,m)$ of Table 4 is defined by $t_4(n,m) = t_3(n,m) - t_1(n,m)$, where $t_i(n,m)$ denotes the (n,m) entry of Table i. In Table 4 as well as in other tables, the sign - expresses meaningless data or times below the measure threshold of the operating system.

Table 5 gives the average times for evaluating one model constraint during the computation of the first (and only) minimal model of $B(n,m)$.

m / n	5	6	7	8	9	10
4	-	-	0.20	0.55	1.22	2.74
5	0.38	2.24	2.27	10.78	16.17	20.81

Table 5: Average CPU times in 10^{-3} seconds for evaluating one clause in $C(n, m, m^n)$ during the computation of the first (and only) minimal model of $B(n, m)$.

The (n, m) entry $t_5(n, m)$ of Table 5 is defined by:

$$t_5(n,m) = \frac{t_3(n,m) - t_1(n,m)}{c_B(n,m) - c_A(n,m)} \times 10^3 = \frac{t_3(n,m) - t_1(n,m)}{m^n - 1} \times 10^3$$

where $t_i(n, m)$ denotes the (n, m) entry of Table i.

Comparing Table 1 and Table 4 shows that the time needed for one evaluation of the model constraints is much less than the time needed for the rest of the computation.

Admittedly, the B benchmark suite might be less meaningful for minimal model generators that, unlike MM-SATCHMO, do not rely on model constraints.

Niemelä's Scheme: The D Benchmark Suite. In [48] an approach to minimal model generation is described and two examples are considered. The D benchmark suite is a generalization of these examples. For nonnegative integers n, m, and k:

$$D(n, m, k) := E(n, m, k) \cup A(n, m)$$

where

$$E(n,m,k) := \bigcup_{j=1}^{k} \{\ \texttt{a_i+1_j :- a_i_j} \mid \texttt{i} = 1, \ldots, n-1\ \}$$

The clauses $E(n, m, k)$ can be seen as k "chains" of implications between literals. These chains express – simple – dependencies between literals, thus conveying – in a rather simple manner – the often more complex literal dependencies present in most practical applications. Arguably, the D benchmark suite is better an approximation of "real life applications" than the A suite.

Following an already mentioned optimization, the clauses in $E(n, m, k)$ are expressed using `:-` instead of `--->`. As discussed in [45], this does not affect the correctness and completeness of the method, since the considered clauses are Horn clauses and their head atoms are not involved in recursion cycles.

The chains considerably reduce the search space, as Tables 6 to 11 show.

n \ m	3	4	5	6	7	8	9	10
3	-	0.01	0.03	0.10	0.15	0.31	0.58	1.02
4	0.02	0.10	0.39	1.34	4.42	14.26	40.86	98.74
5	0.07	0.63	5.19	42.13	229.14	928.12	3 165.38	8 819.52

Table 6: CPU times in seconds for computing all minimal models of $D(n, m, 1)$.

Proposition 34 *The number $d(n, m, k)$ of minimal models of $D(n, m, k)$ $(1 \leq k \leq m)$ is given by the equations:*

$$\begin{aligned} d(1, m, k) &= m \\ d(n+1, m, k) &= k + ((m-k) \times d(n, m, k)) \end{aligned}$$

Proof: SATCHMO and MM-SATCHMO clearly generate m minimal models from a single positive clause of length m, thus $d(1, m, k) = m$. Consider now the set of clauses $D(n+1, m, k) = E(n+1, m, k) \cup A(n+1, m)$ and the "first" clause `true ---> a_1_1 ; a_1_2 ; ... ; a_1_m` of $A(n+1, m)$. 1. If one of the `a_1_j` for `j` $= 1, \ldots, k$ is assigned the truth value "true", so are the atoms `a_h_j` for `h` $= 2, \ldots, n$ and `j` $= 1, \ldots, k$ also assigned the value "true" because of the k "implication chains" in $E(n+1, m, k)$, and the remaining n clauses of $A(n+1, m)$ are all satisfied. Therefore, there are exactly k minimal models of $D(n+1, m, k)$ such that one of the `a_1_j` for `j` $= 1, \ldots, k$ is true. 2. If now for some `j` $= k+1, \ldots, m$, `a_1_j` is true, there are exactly $d(n, m, k)$ minimal models of the remaining clauses in $D(n+1, m, k)$. Since `a_h_l` $\neq$ `a_p_q` for (`h`, `l`) $\neq$ (`p`, `q`), each of these minimal models results in a minimal model of $D(n+1, m, k)$ when extended with the assignment of "true" to `a_1_j`. 3. Thus, $d(n+1, m, k) = k + ((m-k) \times d(n, m, k))$. ∎

Table 7 gives the average times needed for computing one minimal model of $D(n, m, 1)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.24	0.35	0.64	0.58	0.77	0.99	1.24
4	0.65	0.83	1.14	1.72	2.84	5.09	8.73	13.38
5	1.11	1.73	3.80	10.79	24.56	47.33	84.52	132.76

Table 7: Average CPU times in 10^{-3} seconds for computing one minimal model of $D(n, m, 1)$.

The (n, m) entry $t_7(n, m)$ of Table 7 is defined by $t_7(n, m) = \frac{t_6(n,m)}{d(n,m,1)} \times 10^3$ where $t_6(n, m)$ denotes the (n, m) entry of Table 6.

Tables 8 to 11 give the overall and average times per minimal model for $D(n, m, \lfloor \frac{m}{2} \rfloor)$ and $D(n, m, m-1)$. Tables 9 and 11 are obtained from Tables 8 and 10, respectively, like Table 7 is computed from Table 6 by considering the relevant values of $d(n, m, k)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.02	0.04	0.06	0.12	0.18	0.33	0.45
4	0.02	0.06	0.23	0.47	1.42	2.41	6.90	10.70
5	0.07	0.29	1.84	4.38	28.98	57.45	315.45	542.96

Table 8: CPU times in seconds for computing all minimal models of $D(n, m, \lfloor \frac{m}{2} \rfloor)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.71	0.62	0.62	0.69	0.78	0.89	0.98
4	0.65	0.73	0.89	1.00	1.37	1.50	2.36	2.61
5	1.11	1.19	1.80	1.87	4.66	5.13	13.48	14.71

Table 9: Average CPU times in 10^{-3} seconds for computing one minimal model of $D(n, m, \lfloor \frac{m}{2} \rfloor)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.01	0.02	0.04	0.07	0.11	0.16	0.23
4	0.02	0.06	0.13	0.27	0.49	0.86	1.43	2.23
5	0.05	0.28	0.63	1.59	3.51	7.09	13.14	22.88

Table 10: CPU times in seconds for computing all minimal models of $D(n, m, m-1)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.62	0.80	1.11	1.43	1.72	1.97	2.30
4	1.18	1.39	1.46	1.68	1.85	2.11	2.41	2.69
5	1.52	2.26	1.83	2.02	2.25	2.52	2.80	3.10

Table 11: Average CPU times in 10^{-3} seconds for computing one minimal model of $D(n, m, m-1)$.

The run times for the D suite are significantly smaller than for the A suite. Moreover, the average times significantly decrease when the number of "implication chains" increases, showing that MM-SATCHMO well propagates truth values assignments through ("chains" of) implications. Arguably, this is a significant factor of efficiency. Since Niemelä's scheme is a good approximation of "real life examples" the performance of MM-SATCHMO on these examples gives support to the claim, that this rather simple implementation is sufficient for many practical applications.

A Strengthening of Niemelä's Scheme: The F Benchmark Suite. Like the B suite strengthens the A suite by adding model constraints, Niemelä's scheme (or D suite) suite can be strengthened into the F suite as follows:

$$F(n,m,k) := C(n,m,m^n) \cup D(n,m,k) = C(n,m,m^n) \cup E(n,m,k) \cup A(n,m)$$

n \ m	3	4	5	6	7	8	9	10
3	-	0.02	0.03	0.08	0.17	0.33	0.61	1.14
4	0.02	0.11	0.50	2.19	8.81	26.46	67.39	154.71
5	0.12	1.40	16.60	100.38	462.95	1 688.92	5 704.90	15 242.60

Table 12: CPU times in seconds for computing all minimal models of $F(n,m,1)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.01	0.03	0.07	0.15	0.28	0.52	0.84
4	0.02	0.09	0.43	1.55	6.70	17.80	49.20	99.84
5	0.12	0.97	12.69	60.84	303.68	989.93	3 585.84	8 608.44

Table 13: CPU times in seconds for computing all minimal models of $F(n,m,\lfloor \frac{m}{2} \rfloor)$.

n \ m	3	4	5	6	7	8	9	10
3	-	0.01	0.03	0.06	0.10	0.19	0.31	0.50
4	0.02	0.07	0.24	0.94	3.53	9.69	23.17	50.03
5	0.08	0.59	6.06	32.30	134.27	482.85	1 498.11	4 202.49

Table 14: CPU times in seconds for computing all minimal models of $F(n,m,m-1)$.

The comparison of Table 12, 13 and 14 with Table 6, 8, and 10 respectively confirms the observation made with the B suite: Even in presence of a huge number of constraints good run times are achieved. This further supports the claim that MM-SATCHMO, in spite of its simplicity, is a convenient minimal model generator for many applications.

Optimization Potential. Like SATCHMO, MM-SATCHMO emphasizes a principle, not implementation aspects. In implementing SATCHMO and MM-SATCHMO, no attention has been paid to efficiency. The reported run times are therefore noticeable.

In [69, 29] it is shown how natural optimizations dramatically improve the efficiency of SATCHMO. These optimizations consist in (1) computing violated clause instances incrementally, so as to avoid useless repeated computations, (2) specializing the SATCHMO meta-interpreter with respect to the considered set of clauses, so as to avoid the meta-interpretation overhead – a technique called "compilation" in [69, 29] –, (3) a more efficient implementation of complement-splitting, (4) a more efficient search strategy for ensuring fairness, and (5) enhancing the representation language – e.g. with disjunctions in clause bodies. In some cases, gains in efficiency of several orders of magnitudes can be achieved with these techniques.

All these techniques are applicable to MM-SATCHMO, too. The more efficient implementation of complement splitting is especially promising. In contrast, the model constraints upon which MM-SATCHMO relies for avoiding a repeated generation of minimal models do not seem easily amenable to the optimization and compilation techniques investigated in [69, 29]. Nonetheless, these techniques are promising for MM-SACHTMO, since, as observed with the A suite (cf. Tables 1 and 4), the time spent for one evaluation of all model constraints is much smaller than the time needed for the rest of the computation.

First-order Logic vs. Propositional Logic. The examples considered above are all propositional logic examples. For three reasons, examples with first-order variables have not been retained. First, the techniques applied by the approach considered here for restricting the model generation to minimal models do not depend on variables. Second, considering propositional logic examples makes it possible to compare the run times with that of other minimal model generators that do not handle variables. Finally, most problems can be naturally expressed both, without and with first-order variables, and with MM-SATCHMO the representations with variables often yield better run times than the propositional logic representation.

The last claim is conveniently illustrated by the following example. $B(n, m) = C(n, m, m^n) \cup A(n, m)$ includes 99 999 clause in $C(n, m, m^n)$ of the form `false :- Mj` where `Mj` is the Prolog conjunction of the atoms true in one of the first 99 999 minimal models of $A(n, m)$ returned by MM-SATCHMO. With variables, $A(5, 10)$ is conveniently and naturally represented by the five facts `index(1)` , ..., `index(5)` and the clause:

```
index(J) ---> a_1(J) ; a_2(J) ; ... ; a_10(J)
```

With this representation, it is natural to replace the 99 999 clauses of $C(n, m, m^n)$ by the following 9 clauses:

```
false :- a_1(J)
false :- a_2(J)
   ⋮
false :- a_9(J)
```

Indeed, in every minimal model of $A(5, 10)$ except the last one returned by MM-SATCHMO, `a_i(j)` is true for some `i` $\in \{1, \ldots, 5\}$ and `j` $\in \{1, \ldots, 9\}$.

For generating the single minimal model of the propositional logic representation of $B(5, 10)$, MM-SATCHMO takes 15 398.20 seconds (CPU time) – cf. Table 3 – with clauses for `false` declared dynamic and the loading times not considered. With the first-order representation of the same example given above, MM-SATCHMO needs only 27.89 seconds (CPU time) for the same task. Admittedly, this example is an extreme case. In general, the speed up obtainable by changing the representation is less considerable.

Comparison of Performances. Up to a renaming of the propositional variables, $D(4,5,1)$ is identical with the example $\Upsilon \cup S_a$ of [48]. That article reports a run time of "less than 2 seconds" for generating all minimal models of this example with an implementation in ECLiPSe Prolog run "on a SUN Sparc 4" workstation. For the same task, MM-SATCHMO needs 0.39 seconds (cf. Table 6). A second run time reported in [48] is "less than 0.5 seconds" for generating all minimal models of an example denoted $\Upsilon \cup S_a \cup S_b \cup S_c \cup S_d$ which, up to a variable renaming, corresponds to $D(4,5,4)$. For the same task, MM-SATCHMO needs 0.13 seconds (cf. Table 10). Obviously, the method described in [48] and MM-SATCHMO achieve comparable efficiencies on these examples.

In [48] the implementation [21] of a method described in [47] is mentioned without detailed comparisons, because this "implementation was not able to handle very large examples" like $\Upsilon \cup S_a$ or $\Upsilon \cup S_a \cup S_b \cup S_c \cup S_d$. Under "large", not only the number of clauses, but also the number of minimal models is meant. Obviously, the system presented in [21] could not run the benchmark examples considered here, some of which having up to 100 000 clauses, others up to 100 000 minimal models.

A comparison with the performances of DisLog [63, 64], of [50], of [78], or of [14] would not really make sense, because these approaches have not been primarily developed for an efficient generation of minimal Herbrand models. Note that the system DisLog [63, 64] cannot cope with large numbers of minimal models, as considered in this section, and that run times of the algorithms described in the other papers are not available.

6 Conclusion

In this article, positive unit hyperresolution (PUHR) tableaux are defined and their properties investigated. PUHR tableaux formalize the approach to theorem proving of [44, 45]. Then, PUHR tableaux are applied to specifying two procedures for computing the minimal Herbrand models of sets of range restricted clauses. The first minimal model generation procedure performs a depth-first expansion of PUHR (complement) 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 in Prolog of the depth-first minimal model generation procedure in the form of a short Prolog program called MM-SATCHMO is also presented. Its efficiency on extensive benchmarks is reported. The prototype is able to deal with sets of clauses with a very large number of minimal models. Its performances are comparable to the best reported in the literature [48]. MM-SATCHMO has a considerable potential for optimizations like discussed in [69, 29].

As tableaux methods, the proposed approaches enjoy a good degree of efficiency stemming from restricted search spaces, limited applications of expansion rules and the use of matching (without occur-check) rather than full unification. The proposed approaches expand ground tableaux. Since it makes instantiation necessary, this might be considered as a source of inefficiency in a refutation procedure. However, if Herbrand models are to be characterized as sets of ground atoms, as it is considered in this paper, this objection does not apply to a model generation procedure.

As model generation procedures, the approaches to minimal model generation proposed in this paper compare well with those reported in the literature, many of which generate nonminimal models [66, 35, 44, 25, 45, 34, 26, 24, 36, 5, 67, 73, 74, 39, 42, 43, 30, 23, 3, 37, 69, 31, 1, 29, 52, 27, 11]. Compared with approaches based on "blind" model construction then testing for minimality– as e.g. the methods reported in [24, 63, 64] – the approaches proposed here avoid nonminimal model generation altogether. The construction of nonminimal models is aborted as soon as possible, in general before they are fully developed. Also, the methods proposed in this paper are applicable to first-order clauses and not confined to propositional or ground clauses as the algorithms reported in [24, 78, 48]. Note, however, that most of the techniques increasing the efficiency for propositional or ground clauses proposed in e.g. [78, 48] can be incorporated into the algorithms described here. Moreover, the approaches proposed here require no order to be placed on the sequence in which individual atoms are expanded, although, if needed, such an order can be incorporated without substantial changes to the algorithm [78].

Among the limitations of the procedures described in this paper are their applicability only to range restricted and so-called finitary sets of first-order clauses. However, range restrictedness 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. Arguably, 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.

Acknowledgments

The authors thank Norbert Eisinger, Tim Geisler, Heribert Schütz, and the anonymous referees for useful suggestions. 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 1298, 256–272, 1997.

[2] C. Aravindan and P. Baumgartner. A Rational and Efficient Algorithm for View Deletion in Databases. In *Logic Programming – Proc. Int. Logic Programming Symp.*, MIT Press, 1997.

[3] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. Fourth European Workshop on Logic in Artificial Intelligence*, Springer-Verlag, LNCS 1126, 456–459, 1996.

[4] P. Baumgartner, U. Furbach, and I. Niemelä. A Tableau Calculus for Diagnosis Applications. In *Proc. Sixth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Springer-Verlag, LNCS 1227, 1997.

[5] B. Beckert and R. Hähnle. An Improved Method for Adding Equality to Free Variable Semantic Tableaux. In *Proc. Eleventh Int. Conf. on Automated Deduction*. Springer-Verlag, LNCS 607, 507–521, 1992.

[6] F. Bry. Intensional Updates: Abduction via Deduction. In *Proc. Seventh Int. Conf. on Logic Programming,* MIT Press, 561–575, 1990.

[7] F. Bry, N. Eisinger, H. Schütz, and S. Torge. SIC: An Interactive Tool for the Design of Integrity Constraints (System Description). In *Proc. Demo Session, Session of the Sixth Int. Conf. on Extending Database Technology*, 45–46, 1998.

[8] F. Bry and R. Manthey. Detecting Consistency of Database Rules by Adapting Theorem Proving Methods. Technical Report KB-8, ECRC, Munich, 1985.

[9] F. Bry and R. Manthey. Checking Consistency of Database Constraints: A Logical Approach. In *Proc. Twelfth Int. Conf. on Very Large Data Bases*, 1986.

[10] F. Bry and R. Manthey. Proving Finite Satisfiability of Deductive Databases. In *Proc. First Workshop on Computer Science Logic*, Springer-Verlag LNCS 329, 44–55, 1987.

[11] F. Bry and S. Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. Sixth European Workshop on Logics in AI,* Springer LNCS 1489, 1998.

[12] 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 1071, 1996.

[13] R. Caferra and N. Zabel. Building Models by Using Tableaux Extended by Equational Problems. In *Jour. of Logic and Computation*, 3, 3–25, 1993.

[14] W. Chen and D. S. Warren Computation of Stable Models and its Integration with Logical Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 8(5), 742–757, 1996.

[15] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory, *Jour. of the ACM*, 7(3), 201–215, 1960.

[16] M. Denecker and D. Schreye. A Framework for Indeterministic Model Generation With Equality. In *Proc. Conf. on Fifth Generation Computer Systems*, 650–657, 1992.

[17] M. Denecker and D. Schreye. On the Duality of Abduction and Model Generation in a Framework for Model Generation With Equality. *Theoretical Computer Science*, 122, 225–262, 1994.

[18] H.-D. Ebbinghaus, J. Flum, and W. Thomas. Mathematical Logic. Springer-Verlag, 1996.

[19] ECLiPSe – ECRC Common Logic Programming System. User Manual, ECRC, Munich, 1994.

[20] N. Eisinger and T. Geisler. Problem Solving with Model-Generation Approaches based on PUHR Tableaux. In *Proc. Workshop on Problem-solving Methodologies with Automated Deduction, Workshop at the Fifteenth Int. Conf. on Automated Reasoning*, 1998.

[21] R. Emery. Computing Circumscriptive Databases. Master's Thesis, University of Maryland, 1992. Cited in [48].

[22] 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

[23] C. Fermüller and A. Leitsch. Hyperresolution and Automated Model Building. *Jour. of Logic and Computation*, 6(2),173–203, 1996.

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

[25] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, 1987, second edition 1990.

[26] H. Fujita and R. Hasegawa. A Model Generation Theorem Prover in KL1 Using a Ramified Stack Algorithm. In *Proc. Eighth Int. Conf. on Logic Programming*, MIT Press, 1991.

[27] U. Furbach, ed. Tableaux and Connection Calculi. Part I. In *Automated Deduction – A Basis for Applications.* Kluwer Academic Publishers, 1998.

[28] P. Gardenförs. *Knowledge in Flux: Modeling the Dynamic of Epistemic States.* MIT Press, 1988.

[29] T. Geisler, S. Panne, and H. Schütz. Satchmo: The Compiling and Functional Variants. *Jour. Automated Reasoning*, 18(2), 227–236, 1997.

[30] R. Hähnle. Positive Tableaux. Research Report, Computer Science Department, University of Karlsruhe, 1995. Cited in [3].

[31] R. Hasegawa, H. Fujita and M. Koshimura. MGTP: A Model Generation Theorem Prover – Its Advanced Features and Applications. In *Proc. Sixth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Springer-Verlag, LNCS 1227, 1–15, 1997.

[32] R. Hasegawa, K. Inoue, Y. Ohta, and M. Koshimura. Magic Sets to Incorporate Top-Down Inference Into Bottom-Up Theorem Proving. In *Proc. Fourteenth Int. Conf. on Automated Deduction.* Springer-Verlag, LNCS 1249, 176–190, 1997.

[33] L. Herr. E-SATCHMO: Introduction de l'Égalité Dans le Démonstrateur Automatique SATCHMO. Technical Report, ECRC, Munich, 1993. In French.

[34] J. Hintikka. Model Minimization – an Alternative to Circumscription. *Jour. of Automated Reasoning*, 4, 1–13, 1988.

[35] K. M. Hörnig. Generating Small Models of First Order Axioms. *Proc. Sixth German Workshop on Artificial Intelligence*, Springer-Verlag, IFB 47, Springer-Verlag, 1981.

[36] 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*, 400–415, 1992.

[37] M. Kühn. Rigid Hypertableaux. In *KI'97: Advances in Artificial Intelligence, Proc. of Twentyfirst Annual German Conf. on Artificial Intelligence*, Springer-Verlag, LNCS 1303, 1997.

[38] A. Leitsch. *The Resolution Calculus.* Springer-Verlag, 1997.

[39] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule Into Connection Tableau Calculi. *Jour. of Automated Reasoning*, 13(3), 297–338, 1994.

[40] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984, second edition 1987.

[41] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[42] S. Lorenz. A Tableau Prover for Domain Minimization. *Jour. of Automated Reasoning*, 13, 375–390, 1994.

[43] D. Loveland, D. Reed, and D. Wilson. SATCHMORE: SATCHMO With RElevancy. *Jour. of Automated Reasoning*, 14, 325–351, 1995.

[44] 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, IFB 152, 456–459, 1987.

[45] R. Manthey and F. Bry. SATCHMO: a Theorem Prover Implemented in Prolog. In *Proc. Ninth Int. Conf. on Automated Deduction*, Springer-Verlag, LNCS 310, 415–434, 1988.

[46] V. W. Marek and M. Truszczyński. *Nonmonotonic Logic. Context Dependent Reasoning.* Springer-Verlag, 1993.

[47] A. Nerode, R. T. Ng, and V. S. Subrahmanian. Computing Circumscriptive Databases: I. Theory and Algorithms. *Information and Computation*, 116, 58–80, 1995. Cited in [48].

[48] 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 1071, 1996.

[49] I. Niemelä. Implementing Circumscription Using a Tableau Method. In *Proc. Twelfth European Conf. on Artificial Intelligence*, John Wiley & Sons, Ldt, 1996

[50] N. Olivetti. Tableaux and Sequent Calculus for Minimal Entailment. *Jour. of Automated Reasoning*, 9, 99–139, 1992.

[51] M. Paramasivam and D. Plaisted. Automated Deduction Techniques for Classification in Description Logic Systems. *Jour. of Automated Reasoning*, 20(3), 1998.

[52] N. Peltier. Simplifying and Generalizing Formulae in Tableaux. Pruning the Search Space and Building Models. In *Proc. Sixth Workshop on Theorem Proving with Tableaux and Related Methods*, Springer LNCS 1227, 313–327, 1997.

[53] D. Poole. Explanation and Prediction: An Architecture for Default and Abductive Reasoning. *Computational Intelligence*, 5(2), 97–110, 1989.

[54] D. Poole, R. Goebel, and R. Aleliunas. THEORIST: A Logical Reasoning System for Default and Diagnosis. In *The Knowledge Frontier: Essays in the Representation of Knowledge*, N. Cercone and G. McCalla, eds., Springer-Verlag, 1987.

[55] D. Prawitz. A new Improved Proof Procedure. *Theoria*, 26, 102–139, 1960.

[56] P. W. Purdom, Jr. Solving Satisfiability With Less Searching. *IEEE Trans. on Pattern Analysis and Maschine intelligence*, PAMI-6, 4, 510–513, 1984.

[57] A. K. Rajasekar. Semantics for Disjunctive Logic Programs. PhD Thesis, Institute for Advanced Studies and Department of Computer Science, University of Maryland, 1989.

[58] A. K. Rajasekar. Magic Set Evaluation in Disjunctive Databases. Research Report, Computer Science Department, University of Kentucky, 1995.

[59] A. K. Rajasekar and J. Minker. A Fixpoint Semantics for Disjunctive Logic Programs. *Jour. of Logic Programming*, 9(1), 45–74, 1990.

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

[61] R. Reiter. A Theory of Diagnosis From First Principles. *Artificial Intelligence,* 32, 57–95, 1987.

[62] J. A. Robinson. Automatic Deduction With Hyper-Resolution. *Int. Jour. of Computational Mathematics*, 1, 227–234, 1965.

[63] D. Seipel. DisLog - A Disjunctive Deductive Database Prototype. In *Proc. Twelfth Workshop on Logic Programming*, 1997.

[64] D. Seipel. DisLog - A System for Reasoning in Disjunctive Deductive Databases. In *Proc. Int. Workshop on the Deductive Approach to Information Systems and Databases*, 1994.

[65] J. Slaney. Finder (finite domain enumerator): Notes and Guides. Tech. Rep., Australian National University Automated Reasoning Project, Canberra, 1992.

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

[67] M. E. Stickel. Automated Theorem Proving Research in the Fifth Generation Computer Systems Project: Model Generation Theorem Provers. *Future Generation Computer Systems*, 9(2), 143–152, 1993.

[68] M. A. Suchenek. First-Order Syntactic Characterizations of Minimal Entailment, Domain-Minimal Entailment, and Herbrand Entailment. *Jour. of Automated Reasoning*, 10, 237–263, 1993.

[69] H. Schütz and T. Geisler. Efficient Model Generation Through Compilation. *Jour. of Information and Computation*, to appear. Short version in *Proc. Thirteenth Conf. on Automated Deduction*, Springer-Verlag, LNCS 1104, 433–447, 1996.

[70] T. Tammet. Using Resolution for Deciding Solvable Classes and Building Finite Models. In *Proc. Baltic Computer Science Conf.*, Springer-Verlag, LNCS 502, 33–64, 1991.

[71] S. Torge. Überprüfung der Erfüllbarkeit im Endlichen: Ein Verfahren und seine Anwendung. PhD Thesis, Institute for Computer Science, University of Munich, 1998. In German.

[72] M. Winslett. Reasoning About Actions Using a Possible Models Approach. *Proc. Seventh Nat. Conf. on Artificial Intelligence*, 1988.

[73] G. Wrightson, ed. Special Issue on Automated Reasoning With Analytic Tableaux, Part I. *Jour. of Automated Reasoning*, 13(2), 173–281, 1994.

[74] G. Wrightson, ed. Special Issue on Automated Reasoning With Analytic Tableaux, Part II. *Jour. of Automated Reasoning*, 13(3), 283–421, 1994.

[75] A. Yahya. Model Generation in Disjunctive Normal Databases. Tech. Rep. PMS-FB-1996-10, Institute for Computer Science, University of Munich, 1996. http://www.informatik.uni-muenchen.de/pms/publikationen/berichte/PMS-FB-1996-10.ps.gz

[76] A. Yahya. A Goal-Driven Approach to Efficient Query Processing in Disjunctive Databases. Tech. Rep. PMS-FB-1996-12, Institute for Computer Science, University of Munich, 1996. http://www.informatik.uni-muenchen.de/pms/publikationen/ berichte/ PMS-FB-1996-12.ps.gz

[77] 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 1265, 1997.

[78] A. Yahya, J.A. Fernandez, and J. Minker. Ordered Model Trees: A Normal Form for Disjunctive Deductive Databases. *Jour. of Automated Reasoning*, 13(1), 117–144, 1994.

[79] J. Zhang and H. Zhang. SEM: A System for Enumerating Models. In *Proc. International Joint Conference on Artificial Intelligence*, 1995.