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

Ludwig–
Maximilians–
Universität–
München–

Model Generation with Existentially Quantified Variables and Constraints

Slim Abdennadher, Heribert Schütz

In: Proc. Sixth International Conference on Algebraic and Logic Programming, Springer LNCS, 1997.
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-4, September 1997

Model Generation with Existentially Quantified Variables and Constraints

Slim Abdennadher and Heribert Schütz

Universität München, Institut für Informatik, Oettingenstr. 67, D-80538 München
{Slim.Abdennadher|Heribert.Schuetz}@informatik.uni-muenchen.de

Abstract. In this paper we present the CPUHR-tableau calculus, a modification of positive unit hyperresolution (PUHR) tableaux, the calculus underlying the model generator and theorem prover Satchmo. In addition to clausal first order logic, CPUHR tableaux are able to manipulate existentially quantified variables without Skolemization, and they allow to attach constraints to these variables as in constraint logic programming. This extension allows to handle efficiently many realistic model generation problems that cannot be handled by model generators for clausal theories such as PUHR tableaux. In this paper we deal with CPUHR tableaux only for formulas without function symbols other than constants.

1 Introduction

In the last years in the automated deduction community there has been increasing interest in *model generation*, that is, in methods for automatic construction of interpretations that satisfy a given theory. This research direction complements the traditional direction of *theorem proving*. In the area of (disjunctive) logic programming forward-chaining methods typically correspond to model generation while backward-chaining methods correspond to (refutational) theorem proving.

Satchmo [11] has been an early approach to automated model generation. It has originally been developed as a tool for checking the consistency of integrity constraints in relational databases: If Satchmo can generate a (Herbrand) model for a set of integrity constraints, then the integrity constraints are consistent, and can therefore be satisfied by some database instance.

Example 1. Consider, for example, this set of integrity constraints for a database with relations for employees, projects, and assignment of employees to projects:

- Every employee is assigned to at least one project.
- Every project has at least one assigned employee.
- If an employee is assigned to a project, then the employee and the project must exist in the respective relations.

In addition we know that $\texttt{mary}$ is an employee. These conditions can be written as formulas[1]

$$\begin{gathered}
\texttt{employee(E)} \rightarrow \exists \texttt{P}\ \texttt{assigned(E,P)} \\
\texttt{project(P)} \rightarrow \exists \texttt{E}\ \texttt{assigned(E,P)} \\
\texttt{assigned(E,P)} \rightarrow \texttt{employee(E)} \\
\texttt{assigned(E,P)} \rightarrow \texttt{project(P)} \\
\texttt{employee(mary)}
\end{gathered}$$

which are implicitly universally closed.

The first and the second formula in this example contain existentially quantified variables. The third and the fourth formula are "referential integrity constraints", a kind of integrity constraints that appear frequently in relational databases. They would require existential variables as well if we had not restricted `employee` and `project` to a single argument for the sake of simplicity.

Satchmo expects its input to be given in clausal form. So the existentially quantified variables have to be Skolemized:

$$\begin{gathered}
\texttt{employee(E)} \rightarrow \texttt{assigned(E,f(E))} \\
\texttt{project(P)} \rightarrow \texttt{assigned(g(P),P)}
\end{gathered}$$

Unfortunately Satchmo does not terminate when applied to the Skolemized clause set because it enumerates the infinite Herbrand model

$$\begin{aligned}
\{&\texttt{employee(mary)}, \texttt{assigned(mary,f(mary))}, \\
&\texttt{project(f(mary))}, \texttt{assigned(g(f(mary)),f(mary))}, \\
&\texttt{employee(g(f(mary)))}, \texttt{assigned(g(f(mary))),f(g(f(mary)))))}, \\
&\ldots\}.
\end{aligned}$$

In this paper we present CPUHR tableaux, a modification of positive unit hyper resolution (PUHR) tableaux [2], the calculus underlying Satchmo. CPUHR tableaux do not require Skolemization of existential variables. Therefore CPUHR tableaux terminate for many applications where PUHR tableaux do not terminate when applied to the respective Skolemized theory. For the theory in Example 1 the finite set

$$\{\texttt{employee(mary)}, \texttt{assigned(mary,P)}, \texttt{project(P)}\}$$

of atoms is generated, which contains a variable `P`. Every ground instantiation of `P` leads to a finite Herbrand model of the theory. This result has the additional advantage that it is closer to the problem specification than a model containing Skolem functions.

CPUHR tableaux allow also to attach constraints[2] to the existentially quantified variables (which explains the "C" in the name). Intuitively, constraints

---

[1] We adopt the Prolog convention that variable names start with capital letters and predicate and constant symbols start with lowercase letters.

[2] Note that by "constraints" we understand constraints in the sense of constraint logic programming, unless we explicitly speak of "integrity constraints" for databases.

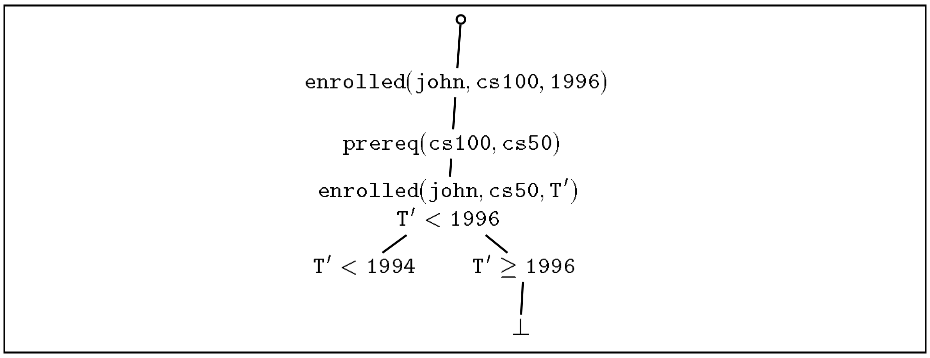

Fig. 1. The tableau generated for Example 2

represent elementary restrictions for and relationships between variables, for example equality or order relationships. They are checked and simplified by a constraint solver that implements a predefined constraint theory.

Example 2. Consider the following informations that we have about a university and a student: If a student $\mathtt{S}$ is enrolled in a course $\mathtt{C}$ at some time $\mathtt{T}$ and $\mathtt{C}$ has another course $\mathtt{C}'$ as a prerequisite, then $\mathtt{S}$ must also be enrolled in $\mathtt{C}'$ at some time $\mathtt{T}'$ before $\mathtt{T}$. $\mathtt{john}$ has only taken courses before $\mathtt{1994}$ and from $\mathtt{1996}$ onward. He has taken course $\mathtt{cs100}$ in $\mathtt{1996}$. $\mathtt{cs100}$ has prerequisite $\mathtt{cs50}$.

1. $\mathtt{enrolled(S, C, T)} \wedge \mathtt{prereq(C, C')} \rightarrow \exists \mathtt{T}'(\mathtt{enrolled(S, C', T')} \wedge \mathtt{T}' < \mathtt{T})$
2. $\mathtt{enrolled(john, C, T)} \rightarrow \mathtt{T} < \mathtt{1994} \vee \mathtt{T} \geq \mathtt{1996}$
3. $\mathtt{enrolled(john, cs100, 1996)}$
4. $\mathtt{prereq(cs100, cs50)}$

Here "$<$" and "$\geq$" are constraint symbols known to the constraint solver.

The tableau generated for this theory is given in Figure 1. First, we introduce facts 3 and 4 from the theory. An instance of rule 1 is activated and we introduce $\mathtt{enrolled(john, cs50, T')}$ and $\mathtt{T}' < \mathtt{1996}$. This activates an instance of rule 2 and we distinguish two cases corresponding to the disjuncts on the right hand side of the implication. In the right case we get a contradiction between two constraints ($\mathtt{T}' < \mathtt{1996}$ and $\mathtt{T}' \geq \mathtt{1996}$) and the branch is closed. We mark this by the symbol "$\perp$".

In the left case no rule application can add new information. For example, rule 2 is not activated by facts 3 and 4 because $\mathtt{T}$ would be instantiated to $\mathtt{1996}$, which would result in a tautology since "$\mathtt{1996} \geq \mathtt{1996}$" is trivially true. The constraint solver collects, combines, and simplifies the constraints in the branch and we get a set of atoms

$$\begin{aligned} A = \{&\mathtt{enrolled(john, cs100, 1996)}, \mathtt{prereq(cs100, cs50)}, \\ &\mathtt{enrolled(john, cs50, T')}\} \end{aligned}$$

and a set of constraints

$$C = \{\mathtt{T}' < \mathtt{1994}\}.$$

We have found out that the conditions of Example 2 are consistent. We have also found out that `john` must have taken `cs50` some time before `1994`. Every valuation for the variable $\mathtt{T}'$ which satisfies C can also be applied to A, which leads (together with an infinite number of atoms for the constraint symbols, which are of no interest here) to a Herbrand model of the theory (and of the constraint theory, of course).

The extension of clausal first order logic by existential variables and constraints allows to handle efficiently many realistic model generation problems that cannot be handled by model generators for clausal theories like PUHR tableaux. Still, one goal in the design of the CPUHR-tableau calculus was to keep much of the "light-weight" flavour of Satchmo and PUHR tableaux, which make their use in (disjunctive) logic programming and deductive databases attractive.

In this paper we deal with CPUHR tableaux only for formulas without function symbols other than constants. Note that this restriction is not as hard as it would be in the absence of existential variables, since (a) we do not need Skolem functions and (b) we will adopt the constraint logic programming approach that a constraint theory does not mainly define properties of functions (constants in our case) by means of equations. We rather leave constants uninterpreted and let the constraint theory define certain predicates.

The paper is organized as follows. The next section gives some definitions and notations which will be used throughout the paper. Section 3 presents the CPUHR-tableau calculus. In section 4 we give soundness and completeness properties for model generation. Section 5 discusses related work. We summarize our contribution and point out some directions for further research in section 6.

2 Preliminaries

We expect the reader to have some basic understanding of logic, tableau calculi, and constraint solving. For the sake of clarity we nevertheless give definitions for some of the notions used in this paper.

We use two disjoint sorts of predicate symbols: *constraint predicates* and *free predicates*. Intuitively, constraint predicates are defined by some constraint theory and handled by an appropriate constraint solver, while free predicates are defined by a user-supplied theory. The CPUHR-tableau calculus tries to compute appropriate interpretations for the free predicates. We call an atomic formula with a constraint predicate an *atomic constraint* and an atomic formula with a free predicate an *atom*.

Throughout the paper we expect some constraint theory CT to be given which has the following properties:

- CT is consistent.
- CT is *ground complete*, that is, for every ground atomic constraint c either $CT \models c$ or $CT \models \neg c$.

– CT does not contain any free predicates.
– CT defines among other constraint predicates equality ("=") and disequality ("$\neq$") as syntactic equality and disequality, using for example Clark's axiomatization.
– There is a constraint solver that implements CT.

Because of the consistence and the ground completeness CT has a single Herbrand model

$$CM := \{c \mid CT \models c\}$$

in the sublanguage without free predicates. As usual, we identify a Herbrand model with the set of ground atomic formulas it satisfies.

We use the notation "$\{v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\}$" for substitutions, which means that the variables $v_1, \ldots, v_n$ are mapped to the terms $t_1, \ldots, t_n$, resp.

For an interpretation I and a variable valuation (sometimes also called "variable assignment") θ we denote the fact that the formula or set of formulas F is satisfied by I and θ as "$I, \theta \models F$". The fact that a closed formula is satisfied by an interpretation I is denoted as "$I \models F$". By "$I, \theta \not\models F$" or "$I \not\models F$" we mean that F is not satisfied by I and θ or by I.

3 The CPUHR-Tableau Calculus

3.1 Syntax

The CPUHR-tableau calculus deals with closed first-order formulas of a certain type, which we call "rules", and it manipulates tableaux of a certain form.

Definition 3. A *rule* is a closed formula of the form

$$\forall\overline{X}(b_1 \wedge \cdots \wedge b_l \rightarrow \exists\overline{Y}(h_1 \vee \cdots \vee h_m \vee c_1 \vee \cdots \vee c_n)),$$

where every b_i and every h_j is an atom, and every c_k is an atomic constraint.

We call the left and right side of the implication the *body* and the *head* of the rule, respectively. We write empty conjunctions and disjunctions as $\top$ and $\perp$, respectively. A rule with an empty body is represented just by its head.

A rule has to satisfy a *range-restriction* condition: Every free variable in the head appears also in the body.

A *specification* is a set of rules.

For brevity we do not write the universal quantifiers in examples, rules are implicitly universally closed.

Example 4. Since conjunctions may not occur in rule heads, we replace condition 1 of Example 2 by the three rules

$$\begin{gathered}\mathtt{enrolled(S,C,T)} \wedge \mathtt{prereq(C,C')} \rightarrow \exists \mathtt{T'}\,\mathtt{aux(S,C',T',T)}\\ \mathtt{aux(S,C',T',T)} \rightarrow \mathtt{enrolled(S,C',T')}\\ \mathtt{aux(S,C',T',T)} \rightarrow \mathtt{T'} < \mathtt{T}\end{gathered}$$

using an auxiliary predicate `aux`.

For the rest of the paper we assume that some specification S is given, for which we are generating models.

We have used the traditional tree representation of a tableau in Figure 1 with open and closed branches. In the formal presentation of the calculus a tableau is given as the set of "open" branches of the tree representation. "Closing" a branch means to remove it from the set. In the representation of branches we separate the atoms and atomic constraints that appear along the corresponding open branch of the tree:

Definition 5. A *branch* is a pair (A, C) where A is a set of atoms, C is a set of atomic constraints. A *CPUHR tableau* is a set of branches.

In contrast to PUHR tableaux, where a branch represents a single Herbrand interpretation, a branch of a CPUHR tableau represents a set of Herbrand interpretations because the variables in the branch are implicitly existentially quantified:

Definition 6. We say that branch (A, C) of a tableau represents the set

$$\begin{aligned} Int(A, C) := \{A\theta \cup CM \mid \theta \text{ is a valuation for the variables in } A \text{ and } C \\ \text{such that } CM, \theta \models C\} \end{aligned}$$

of Herbrand interpretations and that a tableau represents the set

$$Int(T) := \bigcup_{(A,C)\in T} Int(A, C)$$

of Herbrand interpretations.

A formula is *satisfied* by a branch (A, C) if it is satisfied by every interpretation in $Int(A, C)$. It is *violated* by a branch if it is not satisfied by the branch.

Note that $Int(\emptyset) = \emptyset$ and $Int(\{(\emptyset, \emptyset)\}) = Int(\emptyset, \emptyset) = CM$.

3.2 The Inference Rule

CPUHR tableaux for a given specification S are constructed as follows:

- The *initial* tableau $\{(\emptyset, \emptyset)\}$ is a CPUHR tableau for S.
- Further CPUHR tableaux for S are constructed by the following inference rule:

$$\frac{\begin{array}{c} T \text{ is a CPUHR tableau for } S \\ (A, C) \in T \\ E \text{ is a tuple of atoms in } A \\ R \in S \\ R \text{ is applicable to } E \text{ and } (A, C) \end{array}}{T \setminus \{(A, C)\} \cup expand(A, C, E, R) \text{ is a CPUHR tableau for } S}$$

All the complexity of the inference rule is hidden in the notion of applicability and the function *expand*, which are described in detail below. The inference rule is a combination of the PUHR rule and the splitting rule given by Bry and Yahya [2] extended to the more powerful specifications that can be handled by CPUHR tableaux.

In the rest of the paper we will assume that the variables in R are renamed in such a way that no variable in R appears in A or C and that no variable is quantified both universally and existentially in R.

Expansion. The expansion $\mathit{expand}(A, C, E, R)$ of a branch (A, C) with a given rule R and electrons[3] E consists of three steps:

- $\exists$-unification of the rule body with the electrons,
- splitting of the branch according to the disjunction in the rule head, and
- normalization of the constraints.

We explain these steps before we show how they are combined by the definition of the function *expand*.

$\exists$-Unification. We have to "unify" the body of R with the tuple of electrons E. The type of unification that we need is, however, different from the usual unification in resolution calculi because the variables in the electrons are quantified existentially rather than universally. Furthermore note that the electrons may share variables. We call the modified unification *$\exists$-unification.*

As an introductory example consider the rule $\mathtt{p}(\mathtt{Z}, \mathtt{Z}) \rightarrow \mathtt{q}(\mathtt{Z})$ and the branch $(\{\mathtt{p}(\mathtt{X}, \mathtt{a})\}, \emptyset)$. The branch represents the Herbrand interpretations $\{\mathtt{p}(\mathtt{c}, \mathtt{a})\}$ for arbitrary constants $\mathtt{c}$. Only for those interpretations where $\mathtt{c}$ equals $\mathtt{a}$, the rule should be applied. Therefore we distinguish two cases by the constraints "$\mathtt{X} = \mathtt{a}$" and "$\mathtt{X} \neq \mathtt{a}$". In the latter case the rule cannot be applied, while in the former case an instance of the rule should be applied where $\mathtt{Z}$ is instantiated to $\mathtt{a}$.

The $\exists$-unification will in general return a set of solutions rather than a single one. Every solution is a pair (σ, D) where σ is a substitution for the universal variables or "*fail*" and D is a set of atomic constraints. Intuitively, σ is an appropriate instantiation for the universal variables of the rule for those interpretations in $\mathit{Int}(A, C)$ that satisfy D.

The $\exists$-unification proceeds as follows:

- We first check whether the tuple E of electrons has the same number of components as there are atoms in the rule body and whether the predicate symbols and arities of the body atoms coincide with the predicate symbols and arities of the corresponding electrons. If this is not the case, then the $\exists$-unification fails unconditionally (i.e., for every interpretation in $\mathit{Int}(A, C)$) and we return
$$\mathit{ex\_unify}(\mathit{body}(R), E) := \{(\mathit{fail}, \emptyset)\}.$$

[3] As usual in hyperresolution, *electrons* are the "peripheral" formulas involved in a hyperresolution step, that is, all the formulas except for the "central" formula R.

– Otherwise we define a set of equations

$$G := \{t = t' \mid t \text{ is an argument of a body atom and } t' \text{ is the corresponding argument of the corresponding electron.}\}$$

– Every universal variable from the body of R appears on the left hand side of at least one equation in G. We (nondeterministically) choose exactly one equation for every such variable. The chosen equations define a substitution σ for the universal variables. Let G' be the set of remaining equations of G, that is, the equations without universal variables and the equations with universal variables that have not been chosen for σ.
– We now apply σ to G'. The equations in $G'\sigma$ do not contain universal variables any more, but there may be existential variables. We have to deal with those interpretations that satisfy $G'\sigma$ and with those that violate some equation in $G'\sigma$. Therefore we return a solution for each of these cases:

$$\begin{aligned}&\mathit{ex\_unify}(\mathit{body}(R), E) := \\ &\qquad \{(\sigma, G'\sigma)\} \cup \{(\mathit{fail}, \{t \neq t'\}) \mid G'\sigma \text{ contains an equation } t = t'.\}\end{aligned}$$

Note that for every variable valuation θ there is some solution (σ, D) in the result of the ∃-unification (possibly with $\sigma = \mathit{fail}$) such that D is satisfied by CM and θ.

A set D of constraints in a solution returned by the ∃-unification function may be inconsistent or inconsistent with the old constraints in C. Then we might omit the respective solution, but we leave this to the normalization step below.

Example 7. Let R be rule 1 of Example 4 with renamed variables

$$\mathtt{enrolled(A, B, C)} \wedge \mathtt{prereq(B, D)} \rightarrow \exists \mathtt{E}\ \mathtt{aux(A, D, E, C)},$$

and let E be the pair (`enrolled(john, C1, T)`, `prereq(C2, C1)`) of electrons, which means that at some unknown time `T` the student `john` has taken some unknown course `C1`, which is a prerequisite for some other unknown course `C2`. The ∃-unification algorithm computes the following values:

$$\begin{aligned}G &= \{\mathtt{A} = \mathtt{john}, \mathtt{B} = \mathtt{C1}, \mathtt{C} = \mathtt{T}, \mathtt{B} = \mathtt{C2}, \mathtt{D} = \mathtt{C1}\} \\ \sigma &= \{\mathtt{A} \mapsto \mathtt{john}, \mathtt{B} \mapsto \mathtt{C1}, \mathtt{C} \mapsto \mathtt{T}, \mathtt{D} \mapsto \mathtt{C1}\} \\ G' &= \{\mathtt{B} = \mathtt{C2}\} \\ G'\sigma &= \{\mathtt{C1} = \mathtt{C2}\} \\ \mathit{ex\_unify}(\mathit{body}(R), E) &= \{(\sigma, \{\mathtt{C1} = \mathtt{C2}\}), (\mathit{fail}, \{\mathtt{C1} \neq \mathtt{C2}\})\}\end{aligned}$$

We expect that frequently G' is empty or contains only trivially valid equations as in the following example, so that the possible branching that will be introduced by the ∃-unification step (see below) will not matter too much.

Example 8. Let R be rule 2 of Example 2 with renamed variables

$$\texttt{enrolled(john, A, B)} \rightarrow \texttt{B} < \texttt{1994} \vee \texttt{B} \geq \texttt{1996},$$

and let E be the electron $\texttt{enrolled(john, cs30, T)}$. Then the $\exists$-unification algorithm computes

$$\begin{aligned} G &= \{\texttt{john} = \texttt{john}, \texttt{A} = \texttt{cs30}, \texttt{B} = \texttt{T}\} \\ \sigma &= \{\texttt{A} \mapsto \texttt{cs30}, \texttt{B} \mapsto \texttt{T}\} \\ G' &= \{\texttt{john} = \texttt{john}\} \\ \mathit{ex\_unify}(\mathit{body}(R), E) &= \{(\sigma, \{\texttt{john} = \texttt{john}\}), (\mathit{fail}, \{\texttt{john} \neq \texttt{john}\})\} \end{aligned}$$

Splitting. To apply a rule head $\exists \overline{Y}(h_1 \vee \cdots \vee h_m \vee c_1 \vee \cdots \vee c_n)$ in a branch (A, C) we

- split the branch into one branch for every disjunct in the rule head and
- add the disjuncts to the corresponding new branches after applying the substitution σ that has been obtained in the $\exists$-unification step.

This is formalized by the function

$$\begin{aligned} &\mathit{split}(A, C, \exists \overline{Y}(h_1 \vee \cdots \vee h_m \vee c_1 \vee \cdots \vee c_n), \sigma) := \\ &\qquad (A \cup \{h_1\sigma\}, C), \cdots, (A \cup \{h_m\sigma\}, C), (A, C \cup \{c_1\sigma\}), \cdots, (A, C \cup \{c_n\sigma\})\}. \end{aligned}$$

Note that the variables $\overline{Y}$ do not occur in (A, C) and that they are not affected by σ because of the variable renaming mentioned at the beginning of Section 3.2. Note also that the branch is closed if the head is the empty disjunction $\perp$.

Example 9. For the rule head and the substitution σ from Example 8 and the branch $(A, C) = (\{\texttt{enrolled(john, cs30, T)}\}, \{\texttt{T} \leq \texttt{1996}, \texttt{john} = \texttt{john}\})$ the splitting step generates

$$\begin{aligned} \mathit{split}(A, C, \texttt{B} < \texttt{1994} \vee \texttt{B} \geq \texttt{1996}, \sigma) = \{ &(A, \{\texttt{T} \leq \texttt{1996}, \texttt{john} = \texttt{john}, \texttt{T} < \texttt{1994}\}), \\ &(A, \{\texttt{T} \leq \texttt{1996}, \texttt{john} = \texttt{john}, \texttt{T} \geq \texttt{1996}\})\} \end{aligned}$$

Constraint Normalization. We call the constraint solver to close or simplify a tableau branch (A, C):

- The branch is closed if no variable valuation satisfies C together with CM, that is, if $CM \not\models \exists C$.
- Otherwise the branch may be simplified. Let $\overline{X}$ be the common variables of A and C and let $\overline{Y}$ be the local variables of C, that is, those variables that do not appear in A. The constraint solver converts C, X, and Y into a substitution ν for the variables $\overline{X}$ and a set of constraints C' with local variables $\overline{Y'}$ such that

$$CM \models \forall \overline{X}(\exists \overline{Y} \bigwedge C \leftrightarrow \overline{X} = \overline{X}\nu \wedge \exists \overline{Y'} \bigwedge C') \tag{1}$$

  Here "$\overline{X} = \overline{X}\nu$" stands for a conjunction of equations where every variable in $\overline{X}$ is identified with the term to which it is mapped by ν. We typically expect that C' is the same as C or in some way simpler than C.

This leads to the normalization function

$$normalize(A, C) := \begin{cases} \emptyset & \text{if } CM \not\models \exists C \\ \{(A\nu, C')\} & \text{where } \nu \text{ and } C' \text{ satisfy (1), if } CM \models \exists C. \end{cases}$$

Example 10. Consider the branches generated in Example 9. In the first branch the constraint solver might simplify the constraints to "$\mathtt{T} = \mathtt{1996}$" and *normalize* propagates this equality to the atom of the branch:

$$normalize(\{\mathtt{enrolled(john, cs30, T)}\}, \{\mathtt{T} \leq \mathtt{1996}, \mathtt{T} \geq \mathtt{1996}\}) =$$
$$\{(\{\mathtt{enrolled(john, cs30, 1996)}\}, \emptyset)\}$$

In the second branch the constraint solver typically removes the redundant constraint "$\mathtt{T} \leq \mathtt{1996}$":

$$normalize(\{\mathtt{enrolled(john, cs30, T)}\}, \{\mathtt{T} \leq \mathtt{1996}, \mathtt{T} < \mathtt{1994}\}) =$$
$$\{(\{\mathtt{enrolled(john, cs30, T)}\}, \{\mathtt{T} < \mathtt{1994}\})\}$$

Combination of the Expansion Steps. To expand a branch (A, C) with electrons E and rule R we proceed as follows:

- We split (A, C) into one branch for every member (σ, D) of the result of the $\exists$-unification of E and the body of R. We add the constraints in D to the respective branch.
- For a subbranch $(A, C \cup D)$ for which the $\exists$-unification has been successful (i.e., $\sigma \neq \mathit{fail}$) the corresponding instance of the head of R is applied, which means that the subbranch is split and extended again.
- Finally every branch is normalized, which may close some branches.

To formalize the fact that splitting may only be performed if the $\exists$-unification succeeds, we define an auxiliary function *split'*:

$$split'(A, C, H, \sigma) = \begin{cases} \{(A, C)\} & \text{if } \sigma = \mathit{fail} \\ split(A, C, H, \sigma) & \text{otherwise} \end{cases}$$

Now we can define *expand* formally:

$$\begin{aligned} expand(A, C, E, R) = \{(A'', C'') \mid\ & \text{There is a } \sigma\text{, which is a substitution or "}\mathit{fail}\text{"}, \\ & \text{sets } D \text{ and } C' \text{ of atomic constraints, and} \\ & \text{a set } A' \text{ of atoms such that} \\ & (\sigma, D) \in ex\_unify(body(R), E) \text{ and} \\ & (A', C') \in split'(A, C \cup D, head(R), \sigma) \text{ and} \\ & (A'', C'') \in normalize(A', C')\} \end{aligned}$$

Note that there are two possible reasons for branch splitting: Branch splitting can be enforced "implicitly" by the $\exists$-unification and "explicitly" by a disjunctive rule head. There are also two possible reasons for closing a branch: A rule head may be the empty disjunction (i.e., $\perp$) and an introduced constraint may be inconsistent with other constraints.

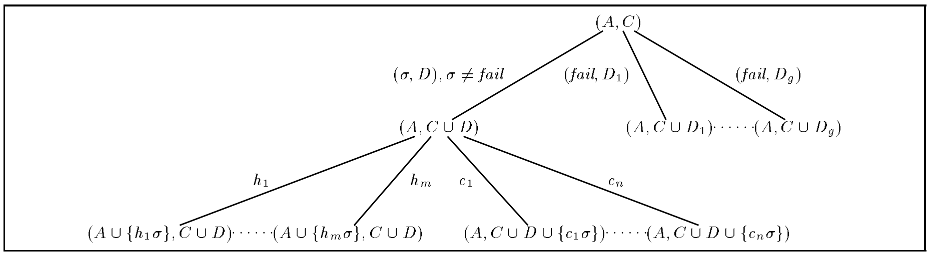

Fig. 2. The branching performed by *ex_unify* and *split*

Figure 2 gives an overview of the branching performed by the $\exists$-unification and splitting steps in the case that one solution of the $\exists$-unification is successful. The normalization step, which would follow every leaf, has been omitted in the figure.

Example 11. Consider the branch (A, C) from Example 9 and the rule R and the electron E from Example 8. Then according to those examples and Example 10 we get

$$\begin{aligned} \mathit{expand}(A, C, E, R) = \{&(\{\mathtt{enrolled(john, cs30, 1996)}\}, \emptyset), \\ &(\{\mathtt{enrolled(john, cs30, T)}\}, \{\mathtt{T} < \mathtt{1994}\})\} \end{aligned}$$

Applicability. It is possible that $\mathit{expand}(A, C, E, R)$ does not really "apply" the rule $R := \forall\overline{X}(b_1 \wedge \cdots \wedge b_l \rightarrow \exists\overline{Y}(h_1 \vee \cdots \vee h_m \vee c_1 \vee \cdots \vee c_n))$:

– If the $\exists$-unification step fails unconditionally (i.e., $\mathit{ex\_unify}(\mathit{body}(R), E) = \{(\mathit{fail}, \emptyset)\}$), then split' does not lead to a modification of the branch (A, C).
– If the $\exists$-unification succeeds for some constraints D (i.e., $\mathit{ex\_unify}(\mathit{body}(R), E)$ contains a solution (σ, D) with $\sigma \neq \mathit{fail}$) but D is not consistent with C (i.e., $CM \not\models \exists \bigwedge(C \cup D)$), then formally split' does apply R, but all the generated branches will be closed by the normalization step.

Furthermore there are cases where R is applied but it does not really add information:

– If the splitting step generates a branch $(A \cup \{h_j\sigma\}, C \cup D)$ where A already contains an atom subsuming $h_j\sigma$, then this branch is essentially the same as the input $(A, C \cup D)$ to the splitting step. Here an atom *subsumes* $h_j\sigma$ if one can instantiate the existential variables $\overline{Y}$ in such a way that $h_j\sigma$ is instantiated to the atom.
– If the splitting step generates a branch $(A, C \cup D \cup \{c_k\sigma\})$ where $c_k\sigma$ is already entailed by $C \cup D$ (i.e., $CM \models \forall(\bigwedge(C \cup D) \rightarrow \exists\overline{Y} c_k\sigma)$, then this branch is essentially also the same as the input $(A, C \cup D)$ to the splitting step.

In all these cases the rule R or the relevant instance of R (i.e., after application of the substitution σ) is already satisfied before the inference step. Since our goal is to generate models for the specification, that is, to satisfy all rules, we avoid such inference steps. Therefore we define that a rule R is applicable to a tuple E of electrons and a branch (A, C) if

- there is a $(\sigma, D) \in \mathit{ex\_unify}(\mathit{body}(R), E)$ with $\sigma \neq \mathit{fail}$,
- $CM \models \exists \bigwedge(C \cup D)$,
- $h_j\sigma\tau \notin A$ for every atom h_j in $\mathit{head}(R)$ and every substitution τ for the variables $\overline{Y}$, and
- $CM \not\models \forall(\bigwedge(C \cup D) \rightarrow \exists\overline{Y} c_k\sigma)$ for every atomic constraint c_k in $\mathit{head}(R)$.

An important effect of this condition is that it avoids trivial nontermination of the calculus by repeated application of a rule to the same electrons.

Example 12. Consider the second of the two branches generated in Example 11 and the rule R and the electron E from Example 8:

$$\begin{aligned}(A, C) = (\{&\texttt{enrolled}(\texttt{john}, \texttt{cs100}, \texttt{1996}), \texttt{prereq}(\texttt{cs100}, \texttt{cs50}),\\ &\texttt{aux}(\texttt{john}, \texttt{cs50}, \texttt{T}', \texttt{1996})\},\\ &\emptyset).\end{aligned}$$

The first rule of Example 4 is not applicable to the electrons $(\texttt{enrolled}(\texttt{john}, \texttt{cs100}, \texttt{1996}), \texttt{prereq}(\texttt{cs100}, \texttt{cs50}))$ and the branch (A, C) because $\texttt{aux}(\texttt{john}, \texttt{cs50}, \texttt{T}'', \texttt{1996})$ is subsumed by the branch.

The second rule of Example 2 is not applicable to the electron $\texttt{enrolled}(\texttt{john}, \texttt{cs100}, \texttt{1996})$ and the branch (A, C) because the constraint $\texttt{1996} \geq \texttt{1996}$ is trivially entailed by any set of constraints.

The following lemma essentially says that the applicability condition is not too restrictive for model generation, because a rule can be applied to a branch if it is not (yet) satisfied by the branch.

Lemma 13. *Let R be a rule that is violated by a branch (A, C). Then there is a tuple E of electrons in A such that R is applicable to E and (A, C).*

Proof. (Sketch) Let R be

$$\forall\overline{X}(b_1 \wedge \cdots \wedge b_l \rightarrow \exists\overline{Y}(h_1 \vee \cdots \vee h_m \vee c_1 \vee \cdots \vee c_n)).$$

There is an interpretation $I \in \mathit{Int}(A, C)$ with $I \not\models R$. So there must be a valuation ρ for the variables $\overline{X}$ with

- $I, \rho \models b_i$ for $1 \leq i \leq l$,
- $I, \rho \not\models \exists\overline{Y} h_j$ for $1 \leq j \leq m$, and
- $I, \rho \not\models \exists\overline{Y} c_k$ for $1 \leq k \leq n$.

According to the definition of $Int(A, C)$ there must be a valuation θ for the variables in (A, C) such that $I = A\theta \cup CM$ and $CM, \theta \models C$.

From the first property given for ρ we can conclude that the $b_i\rho$ occur in I and, since they are atoms rather than constraints, that there are atoms $a_i \in A$ such that $b_i\rho = a_i\theta$. We choose $E := (a_1, \ldots, a_l)$. This choice and the properties of ρ and θ ensure that the four conditions for the applicability of R to E and (A, C) hold. □

4 Soundness and Completeness

We now show that CPUHR tableaux have some desirable properties for model generation.

Definition 14. A branch (A, C) is *saturated* if there is no rule $R \in S$ and no tuple E of electrons in A such that R is applicable to E and (A, C). A tableau is *saturated* if its elements are saturated.[4]

Now the following theorem is an immediate consequence of Lemma 13:

Theorem 15 (Model Soundness). *Let T be a saturated tableau for a specification S and let I be a Herbrand interpretation in $Int(T)$. Then I is a model of S.*

Note that it is not obvious whether it is always possible to construct a saturated tableau. Since saturation means that no rule is applicable, we can simply try to apply all applicable rules with a fair strategy until no more rule can be applied. This works in many cases, but a proof that this will always terminate eventually would have to pose additional requirements on the constraint theory CT.

Theorem 16 (Weak Model Completeness). *Let T be a tableau for the specification S and let M be a Herbrand model of S and CT. Then there is an interpretation $I \in Int(T)$ such that $I \subseteq M$.*

Proof. (By induction on the derivation of T.)

For the initial tableau $\{(\emptyset, \emptyset)\}$ we choose $I := \emptyset \in \{\emptyset\} = Int(\{(\emptyset, \emptyset)\})$, which is trivially a subset of M.

Now let T be a tableau with an interpretation $I \in Int(T)$ with $I \subseteq M$ and let T' be derived by an inference step from T. We show that there is an interpretation $I' \in Int(T')$ with $I' \subseteq M$ (ignoring the constraint normalization steps because they obviously do not change the set of represented interpretations):

I is represented by some branch (A, C) of T. If this is not the branch that is expanded in the step from T to T', then (A, C) is also a branch of T'. In this case we choose $I' := I$ and are done.

[4] Saturation depends like applicability on the given specification and the constraint theory.

Otherwise T' contains the branches $expand(A, C, E, R)$ for some electrons E in A and some rule R in S. There is a valuation θ for the variables in (A, C) such that $I = A\theta \cup CM$ and $CM, \theta \models C$. For one of the pairs (σ, D) returned by the $\exists$-unification the constraints D are also satisfied by CM and θ. It follows that $I \in Int(A, C \cup D)$.

If the corresponding σ is "*fail*", then T' contains the branch $(A, C \cup D)$. In this case we also choose $I' := I$ and are done.

Otherwise, if the corresponding σ is not "*fail*", then the body atoms of $R\sigma\theta$ (R with the universal variables instantiated according to $\sigma\theta$) are equal to the corresponding electrons a_i after application of θ. Since the electrons appear in A, the atoms $a_i\theta$ appear in I and therefore in M.

Since M is a model of the specification, it also satisfies R and also $R\sigma\theta$. According to the previous considerations it satisfies the body of $R\sigma\theta$ and therefore also the head. So there is a valuation θ' that maps the variables in A and C in the same way as θ and the existential variables in R in such a way that $h_j\sigma\theta' \in M$ for some head atom h_j of R or $c_k\sigma\theta' \in M$ for some head constraint c_k of R.

In the former case we choose $I' := A\theta' \cup \{h_j\sigma\theta'\} \cup CM$. We have $I' = I \cup \{h_j\sigma\theta'\} \subseteq M$ and $I' \in Int(A \cup \{h_j\sigma\}, C \cup D)$, where $(A \cup \{h_j\sigma\}, C \cup D)$ is a branch of T'.

In the latter case we choose $I' := A\theta' \cup CM$. We have $I' = I \subseteq M$ and $I' \in Int(A, C \cup D \cup \{c_k\sigma\})$, where $(A, C \cup D \cup \{c_k\sigma\})$ is a branch of T'. $\square$

We call this theorem "weak" because it does not say that every Herbrand model can be constructed, but only that for every Herbrand model a subset can be constructed. This is not a problem, since we usually want to construct minimal rather than arbitrary Herbrand models. With saturated tableaux we get the following completeness result for minimal Herbrand models:

Theorem 17 (Minimal Model Completeness). *Let T be a saturated tableau for the specification S, and let M be a minimal Herbrand model of $S \cup CT$. Then $M \in Int(T)$.*

Proof. By Theorem 16, there is an Interpretation $I \in Int(T)$ such that $I \subseteq M$. Since T is saturated, by Theorem 15 I is a model of S, and since I interprets constraints in the same way as CM, it is a model of CT. Since M is a minimal model of $S \cup CT$ it follows that $I = M$. $\square$

5 Related Work

Without special care, Skolem functions for existentially quantified variables frequently lead to an explosion of the generated models. The introduction of Skolem constants at runtime by the δ-rule of tableau calculi [13] has a similar effect. This behaviour can sometimes be avoided

– either by a modification of the δ-rule as proposed in [3,1], and in [8] under the name δ^*-rule, which tries to reuse old constants before a new constant is introduced,[5] or
– by mapping different ground terms to a single member of the universe (according to an equational theory), which means that a non-Herbrand model is generated.

In CPUHR tableaux existential variables are not Skolemized at all. Furthermore we have made the choice that the constraint theory implemented by the constraint solver essentially defines the interpretations of predicates rather than that of function symbols.

This allows to use a fixed Herbrand domain. We think that for many applications (e.g. database applications, where the set of objects in the real world is fixed) such a domain is most appropriate and allows easy modelling. (Still, different applications might prefer different approaches and some of the three approaches above could be mapped to other ones.)

Kirchner discusses in [9] the use of constraints in automated deduction. She presents three approaches dealing with constraints: Expression of strategies using constraints, schematization of complex unification problems through constraints and the incorporation of built-in theories in a deduction process. The work described in the present paper gives another advantage of the use of constraints especially in model generation: CPUHR-tableau calculus allows to handle efficiently many realistic problems that cannot be handled by model generators for clausal theories like PUHR tableaux.

Other approaches for model generation involving constraints in the literature work with equational constraints:

– Satchmo has been extended to handle equations in rule heads [6]. Equations are added to an equational theory for the current branch at runtime.
– Caferra et al. [5,4] use equality and disequality constraints for model generation. Both their calculus and their use of constraints is quite different to ours: It attaches constraints to universally rather than to existentially quantified variables, and it is mainly based on resolution rather than tableaux.

Constraint Handling Rules (CHR, [7]) is a high-level language for the implementation of constraint solvers. One type of rules in CHR, the "propagation rules" can be augmented by disjunction and are then similar to the rules handled in this paper.

The operational semantics of these rules differs, however, from CPUHR tableaux: Instead of $\exists$-unification it just uses matching.[6] This does never lead to branching or impose new constraints on the existential variables. Furthermore, the mechanism that avoids repeated application of a rule to the same electrons is not comparable to the one used in CPUHR tableaux.

[5] Similar techniques have been used in extensions of Satchmo [6].
[6] In our terminology: Unification succeeds only if G' is already satisfied.

CHR can be used for the implementation of CPUHR tableaux. In fact the idea of combining features of CHR and Satchmo have led to the work presented in this paper.

6 Conclusion and Future Work

In this paper, we have outlined two extensions to the PUHR-tableau calculus. In addition to clausal first order logic, CPUHR tableaux are able to manipulate existentially quantified variables without Skolemization, and they allow to attach constraints to these variables as in constraint logic programming. We have also given certain soundness and completeness properties of the CPUHR-tableau calculus with respect to model generation.

The CPUHR-tableau calculus terminates in many cases where the PUHR-tableau calculus does not terminate when applied to the respective Skolemized specification. Therefore the extensions allow to handle efficiently many realistic model generation problems that cannot be handled by model generators for clausal theories like PUHR tableaux.

Following the considerations in the previous section, our approach to dealing with existential variables and constraints may also be interesting for tableau calculi other than PUHR tableaux.

Our approach can be seen as an extension of disjunctive logic programming with forward chaining [10] by constraints. The requirements for a constraint solver are essentially the same as those posed by traditional constraint logic programming systems with backward chaining.

Interesting directions for future work include

- integration of function symbols in the language,
- optimization possibilities that arise with constraint solvers that can handle certain disjunctions of constraints without case splitting,
- efficient implementation of the calculus, possibly based on constraint handling rules [7] or on the compiling implementation of Satchmo [12],
- determination of properties of constraint theories that guarantee that saturated tableaux can be generated or at least approximated, and
- a description of the first-order theories that can be transformed into specifications for CPUHR tableaux (as in Example 4) and the development of transformation algorithms.

Acknowledgements

We thank Thom Frühwirth for helpful comments on an earlier draft of this paper. The support for the second author by the Bayerischer Habilitations-Förderpreis is appreciated.

References

1. F. Bry and R. Manthey. Checking consistency of database constraints: A logical basis. In *12th Int. Conf. on Very Large Data Bases (VLDB)*, Kyoto, Japan, 1986.
2. F. Bry and A. Yahya. Minimal model generation with positive unit hyper-resolution tableaux. In *5th Workshop on Theorem Proving with Tableaux and Related Methods*, Springer LNAI, 1996.
3. F. Bry. Proving finite satisfiability of deductive databases. In *Proc. of the Conference Logic and Computer Science*, Karlsruhe, Germany, 1987. Springer-Verlag.
4. R. Caferra and N. Peltier. Model building and interactive theory discovery. In *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods 95*, Springer LNAI 918, pages 154–168, 1995.
5. R. Caferra and N. Zabel. Extending resolution for model construction. In *Logics in AI: European Workshop JELIA '90*, Springer LNAI 478, pages 153–169, 1991.
6. M. Denecker and D. De Schreye. On the duality of abduction and model generation in a framework for model generation with equality. *Journal of Theoretical Computer Science*, 1994.
7. T. Frühwirth. Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer-Verlag, 1995.
8. J. Hintikka. Model minimization – an alternative to circumscription. *Journal of Automated Reasoning*, 4(1):1–14, Mar. 1988.
9. H. Kirchner. On the use of constraints in automated deduction. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer, 1995. (Châtillon-sur-Seine Spring School, France, May 1994).
10. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.
11. R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In *9th Int. Conf. on Automated Deduction (CADE)*, Springer LNCS 310, pages 415–434, 1988.
12. H. Schütz and T. Geisler. Efficient model generation through compilation. In M. McRobbie and J. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in Lecture Notes in Artificial Intelligence, pages 433–447. Springer-Verlag, 1996.
13. R. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.