Calculi for Disjunctive Logic Programming

Peter Baumgartner and Ulrich Furbach
Universität Koblenz, Institut für Informatik
Rheinau 1, 56075 Koblenz
Germany

E-mail: {peter,uli}@informatik.uni-koblenz.de

Abstract

In this paper we investigate relationships between top-down and bottom-up approaches to computation with disjunctive logic programs (DLPs). The bottom-up calculus considered, hyper tableaux, is depicted in its ground version and its relation to fixed point approaches from the literature is investigated. For the top-down calculus we use restart model elimination (RME) and show as our main result that hyper tableaux provide a bottom-up semantics for it. This generalizes the well-known result linking the T-operator to SLD-resolution for *definite* programs towards *disjunctive* programs. Furthermore we discuss that hyper tableaux can be seen as an extension of SLO-resolution.

Keywords: Disjunctive Logic Programming, Fixpoint Semantics, SLO

1 Introduction

For disjunctive logic programs (DLPs) there are several proposals for defining interpreters, like the nearHorn-Prolog family [Lov87], SLI-Resolution [LMR92], SLO-Resolution [Raj89], model tree construction [FM91] or restart model elimination (RME) [BF94a, BFS95]. There have also been different approaches to assign least fixpoints to DLPs, like the state semantics [MR90], a semantics based on model trees [FM91] and approaches to give a fixed point semantics to special interpreters, e.g. in [Fur92, Dec91, RS90].

This paper is concerned with relationships among these approaches.

In previous work the authors introduced the family of RME calculi as goal oriented interpreters for positive disjunctive logic programs [BF94a]; more recently, we investigated variants of RME for computing *answers* to queries for DLPs [BFS95]. RME is related to Plaisted's MPRF [Pla88] and Loveland's nearHorn Prolog [Lov87]. The idea throughout these calculi is to enter a clause $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ only through one of the head literals $A_1, \ldots, A_m$, but never through one of the body literals $B_1, \ldots, B_n$. Thus, a natural procedural reading of clauses is better supported than in e.g. model elimination [Lov78].

RME can be implemented by using the PTTP [Sti90] technique. This offers the advantage that in case of Horn programs the procedure *is the underlying PROLOG* system. Only the non-Horn part of a disjunctive logic program is treated by a compiler such that at run time a PROLOG program is executed by an efficient PROLOG system (for details see [BF94b]).

Recently a bottom-up proof procedure for non-ground positive DLPs, the hyper tableaux calculus, was presented in [BFN96]. The present paper shows that this

proof procedure can be understood as a direct implementation of the most prominent fixpoint iteration techniques. Since it suffices for our purposes to restrict to the ground case of Hyper tableaux, and Hyper tableaux coincide in this case with the well-known SATCHMO procedure [MB88], our results apply to SATCHMO as well.

In Section 2 we summarize the bottom-up ground hyper tableaux calculus from [BFN96] and in the subsequent Section 3 we compare this calculus to fixpoint iteration techniques from [MR90] and [FM91].

In Section 4 we will show how a hyper tableaux refutation can be transformed into a RME refutation. This result links the bottom-up to a top-down semantics for DLPs, and thus generalizes the standard result in [Llo87] saying that any finite iteration of the T-operator for *definite* programs can be simulated top-town in a SLD-refutation. In Section 5 we relate SLO-Resolution to Hyper tableaux, and conclude that Hyper tableaux are more general.

Preliminaries

Clauses, i.e. multisets of literals, are usually written as the disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ or as an implication $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ ($m \geq 0$, $n \geq 0$). A clause with $m \geq 1$ is also called a *program clause*, and a clause with $m = 0$ is called a *negative* clause. A *ground* clause or literal contains no variables. With $\overline{L}$ we denote the complement of a literal L. Two literals L and K are *complementary* if $\overline{L} = K$. As usual, an *interpretation* for an (implicitly) given signature is always represented by the set of atoms being *true* in it. A clause C' is *the smallest factor of* clause C iff C' is the shortest subclause of C such that $C \equiv C'$.

We consider finite ordered trees T where every node is labeled with a literal, except the root. Such trees are also called *tableaux*. The labeling function is denoted by λ_T, or simply λ, but in the sequel we will often confuse nodes with their labels. A *branch* of a tableau T is a sequence $N_0, \ldots, N_n$ ($n \geq 0$) of nodes in T such that N_0 is the root of T, N_i is the immediate predecessor of N_{i+1} for $0 \leq i < n$, and N_n is a leaf of T. A branch $b = N_0, \ldots, N_n$ is called *regular* iff $\lambda(N_i) \neq \lambda(N_j)$ for $1 \leq i, j \leq n$ and $i \neq j$, otherwise it is called *irregular*. A tableau is *regular* iff each of its branches is regular, otherwise it is *irregular*. The set of *branch literals* of b is $\mathrm{lit}(b) = \{\lambda(N_1), \ldots, \lambda(N_n)\}$. We find it convenient to use a branch in place where a literal set is required, and mean its branch literals. For instance, we will write expressions like $A \in b$ instead of $A \in \mathrm{lit}(b)$. A clause C is called a *tableau clause (in T)* iff there is a node N in T with $\lambda(N_1) \vee \cdots \vee \lambda(N_n) = C$, where $\{N_1, \ldots, N_n\}$ are all children of N. By $T\delta$ we mean the tableaux T' which results from T by updating the labeling function such that $\lambda_{T'}(N) = (\lambda_T(N))\delta$, where δ is some substitution (i.e. we apply δ to the labels).

A *selection function* is a total function f which maps an open tableau to one of its open branches. If $f(T) = b$ we also say that *b is selected in T by f*. Fortunately, there is no restriction on which selection function to use. For instance, one can use a selection function which always selects the "leftmost" branch.

2 Hyper Tableaux

In [BFN96] we introduced a variant of clausal normal form tableaux called "hyper tableaux". Hyper tableaux keep many desirable features of analytic tableaux (structure of proofs, reading off models in special cases) while taking advantage of central ideas from (positive) hyper resolution. In the ground case, hyper tableaux coincide with the well-known SATCHMO [MB88] procedure; for the first-order case, hyper tableaux have significant advantages (see [BFN96]). For the purposes of the present paper, however, where we use hyper tableaux to model a semantics of positive disjunctive programs, it is sufficient to treat the ground case only. A top-down proof procedure, which is able to handle the first order case and, hence, to compute answers is given later in Section 4.

In order to make the present paper self-contained we will recall a simplified ground version of the calculus. For the rest of this paper $\mathcal{S}$ always denotes a possibly infinite[1] ground clause set, unless stated otherwise; $\mathcal{S}$ is also referred to as the *input clause set*.

Definition 2.1 (Hyper Tableaux)
Let f be a selection function. We consider tableaux where each branch is labeled as either "open" or "closed". *Hyper tableaux* for $\mathcal{S}$ are inductively defined as follows[2]:

Initialization Step: The empty tree, consisting of the root node only, is a hyper tableau for $\mathcal{S}$. Its single branch is labeled as "open".

Hyper Extension Step: If

1. T is an open hyper tableau for $\mathcal{S}$, $f(T) = b$ (i.e. b is the open branch selected in T by f), and
2. $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is a clause from $\mathcal{S}$ ($m \geq 0$, $n \geq 0$), called *extending clause* in this context, and
3. $\{B_1, \ldots, B_n\} \subseteq b$ (referred to as *hyper condition*)

then the tree T' is a hyper tableau for $\mathcal{S}$, where T' is obtained from T by *extension of b by C*: replace b in T by the *new* branches

$$(b, A_1) \ldots, (b, A_m), (b, \neg B_1) \ldots, (b, \neg B_n)$$

and then label every new branch $(b, A_1) \ldots, (b, A_m)$ with positive leaf as "open", and label every new branch $(b, \neg B_1) \ldots, (b, \neg B_n)$ with negative leaf as "closed".

We will write the fact that T' can be obtained from T by a hyper extension step in the way defined as $T \vdash_{b,C} T'$, and say that C is *applicable* to b (or T). Note that the selection function does not appear explicitly in this relation; instead we prefer to let f be given implicitly by the context.

A hyper tableau is *closed* if each of its branches is closed, otherwise it is *open*. □

[1] The ability to handle *in*finite sets of clauses allows easy treatment of the first-order case.

[2] The inductive definition will be such that a branch is closed iff it contains a pair of complementary literals.

The hyper condition of an extension expresses that *all* (which are possibly zero) body literals have to be satisfied by the branch to be extended. This similarity to hyper *resolution* coined the name "hyper tableaux".

The central property of an open branch b is that it can be mapped to an interpretation in the usual way, i.e. by taking the positive literals of b as *true* and all others as *false*; for infinite derivations we take the chain limit of the increasing branches. Together with an appropriate *fairness* notion for derivations (roughly: at least one open branch has to be expanded as long as possible without violating regularity) we get the completeness of hyper tableaux (see again [BFN96]).

Definition 2.2 (Hyper Tableaux Derivation)
Let f be a selection function. A (possible infinite) sequence $T_1, \ldots, T_n, \ldots$ of hyper tableaux for $\mathcal{S}$ is called a *(hyper tableaux) derivation from* $\mathcal{S}$ iff T_1 is obtained by an initialization step, and for $i > 1$, $T_{i-1} \vdash_{b_{i-1},C_{i-1}} T_i$ for some clause $C_{i-1} \in \mathcal{S}$. This is also written as

$$T_1 \vdash_{b_1,C_1} T_2 \cdots T_n \vdash_{b_n,C_n} T_{n+1} \cdots$$

A derivation is called *regular* iff every tableau in the derivation is regular (cf. Section. 1), otherwise it is *irregular*. A derivation is called a *(hyper tableaux) refutation* if it contains a closed tableau. □

Note that extension steps are no longer applicable to a closed hyper tableaux.

Figure 1 shows an example refutation. This example also demonstrates that hyper tableaux handle more than one negative clause. By this it is possible to have integrity constraints in the input clause set, and not just program clauses.

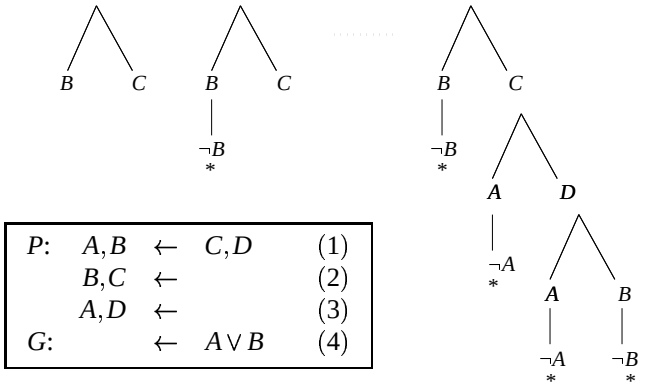


Figure 1: A hyper tableau for P and G given in the figure. Notice that the formula G stands for the two clauses $\leftarrow A$ and $\leftarrow B$. For simplicity of presentation, negative leaf nodes stemming from program clauses are not drawn.

3 States and Model Trees

In this section we relate hyper tableaux to the fixpoint semantics.

3.1 States

We summarize state semantics from [LMR92]. Let $\mathcal{S}^+$ be a possibly infinite set of ground program clauses.

DHB, the disjunctive Herbrand base for $\mathcal{S}^+$ is the set of disjunctions that can be formed by atoms of the Herbrand base of $\mathcal{S}^+$. The transformation $\Gamma : 2^{DHB} \to 2^{DHB}$ is given by[3]

$$\Gamma(D) = \{C \in DHB \mid C' \leftarrow B_1, \ldots, B_n \in \mathcal{S}^+, \quad \forall 1 \leq i \leq n : B_i \vee C_i \in D, \\ C'' = C_1 \vee \ldots \vee C_n \vee C', \quad C \text{ is the smallest factor of } C''\}$$

A *state* for $\mathcal{S}^+$ is a subset of *DHB*. An *expanded state* ST for $\mathcal{S}^+$ is a state, such that $ST = exp(ST)$, where

$$exp(ST) = \{C \in DHB \mid C \in ST \text{ or } \exists C' \in ST : C' \text{ is a subclause of } C\}$$

In [LMR92] it is shown that the operator Γ is continuous, hence its least fixpoint exists and the *lfp*-Operator yields $\Gamma\uparrow\omega$.

We do not introduce model states explicitly, moreover we use their characterization as fixpoints and logical consequences:

Theorem 3.1 (Lobo et al. 92)
Let $\mathcal{S}^+$ be a set of program clauses and $C \in DHB$. Then $\mathcal{S}^+ \models C$ iff $C \in exp(lfp(\Gamma))$.

Definition 3.2 (Cut of T)
Let T be a hyper tableau for $\mathcal{S}$. A clause C is a *cut of T* iff

(1) $\forall b$ open branch of T : $\exists N \in b$: $\exists L \in C$ such that $\lambda(N) = L$, and
(2) $\forall L \in C$: $\exists b$ open branch of T : $\exists N \in b$: $\lambda(N) = L$.

□

That is, in order to get a cut we pick a literal from every open branch.

Now assume that $\mathcal{S}$ is in *goal normal form*. By this we mean the transformation of every clause of the form $\leftarrow B_1, \ldots, B_n$ into $G \leftarrow B_1, \ldots, B_n$, where G is a new predicate symbol, and furthermore adding the clause $\neg G$ as the only goal. By $\mathcal{S}^+$ we denote the set $\mathcal{S}$ without the purely negative clauses. Then $\mathcal{S}^+$ consists of program clauses only and consequently is satisfiable.

The following lemma relates hyper tableaux derivations to Γ-iterations.

Lemma 3.3
For every i and $C \in \Gamma\uparrow i$ there is a hyper tableau T such that there is a cut C' of T were C is the smallest factor of C'.

[3] Obviously, this operator is dependent of the program $\mathcal{S}^+$; we assume this will be clear from the context.

Proof. Induction on i.

Induction start $i = 0$: The set $\Gamma\uparrow 0$ contains all smallest factors of (disjunctive) facts from $\mathcal{S}^+$. Let $A_1, \ldots, A_n \leftarrow$ be such a fact, then construct a hyper tableau with an initial step and a hyper extension step using this fact. The cut $A_1, \ldots, A_n \leftarrow$ of this tableau has the desired property.

Induction step $i \to i+1$: If $C \in \Gamma\uparrow i+1$, we know by definition of Γ that C is the smallest factor of $C_1 \vee \ldots \vee C_n \vee C'$, such that $C' \leftarrow B_1, \ldots, B_n \in \mathcal{S}^+$ and $\{B_1 \vee C_1, \ldots, B_n \vee C_n\} \subseteq \Gamma\uparrow i$. The induction hypothesis gives us that there exist hyper tableaux $T_1, \ldots, T_n$, such that $B_j \vee C_j$ is the smallest factor of a cut $B_j \vee C_j'$ of T_j.

All we have to do is to link these tableaux together to one hyper tableau: Select a branch b_j from tableau T_j which contains the literal B_j. Take the leaf N_j of this branch and use it as the new root of the tableau T_{j+1}. The result is again a hyper tableau. If this linking is done for j from 1 to n we get with T_{n+1} a hyper tableau which contains a branch with literals $B_1, \ldots, B_n$. Hence, the clause $C' \leftarrow B_1, \ldots, B_n$ is applicable, and in the resulting tableau there is a cut $C' \vee C_1' \vee \cdots C_n'$ with smallest factor C. Q.E.D.

As an example, assume the following set of clauses $\mathcal{S}^+ = \{b \vee c, a \leftarrow b, a \leftarrow c\}$. There is a hyper tableau which consists of the two branches with $\{b, a\}$ and $\{c, a\}$ as sets of labels of its nodes. A set of cuts of this tree is $\{a \vee a, a \vee b, b \vee c, a \vee c\}$. Note that there is no sequence assumed in which the literals from different branches have to occur within a cut. The iteration of the Γ-operator gives $\Gamma\uparrow 0 = \{b \vee c\}$, $\Gamma\uparrow 1 = \{b \vee c, a \vee c, a \vee b\}$ and $\Gamma\uparrow 2 = \{b \vee c, a \vee c, a \vee b, a\} = \Gamma\uparrow \omega$.

Having this close relation between hyper tableaux and the fixpoint iteration over states we can use this result to prove completeness of ground hyper tableaux. Note that a proof for full first order clauses is given from scratch in [BFN96], which includes a fairness consideration. Here we only want to establish the close relationship between the approaches.

Theorem 3.4 (Completeness of Hyper Tableaux)
For every unsatisfiable ground clause set $\mathcal{S}$ in goal normal form there is a closed hyper tableau for $\mathcal{S}$.

Proof. Clearly, $\mathcal{S}^+ \models G$. From Theorem 3.1 we learn that $G \in exp(lfp(\Gamma))$. Since G is an atom it must be contained in $lfp(\Gamma)$ alone. Since Γ is continuous, we can apply a standard result from fixpoint theory to conclude $lfp(\Gamma) = \Gamma\uparrow \omega$. Hence there is an i such that $G \in \Gamma\uparrow i$.

Lemma 3.3 gives us the existence of a hyper tableau T with a cut C', such that G is the smallest factor of C'. Hence C' has the form $G \vee \ldots \vee G$; in other words every branch of the tableau contains the literal G. This tableau can be closed by using the goal clause $\neg G$. Q.E.D.

3.2 Model Trees

The other approach we want to relate hyper tableaux to, is that of bottom-up evaluation of disjunctive deductive databases. In [FM91] and [Fur92] a bottom-up conse-

quence operator Γ^M for disjunctive deductive databases is given which acts on sets of interpretations, thus yielding models for the given set of clauses. In [SMR95] this approach is related to the consequence operator on states which we discussed above. Fernandez and Minker also introduce model trees as a calculus to compute this operator, this is done in detail in [LMR92]. We will demonstrate that this is related closely to the hyper tableaux calculus.

The consequence operator Γ^M over sets of Herbrand interpretations is given by

$\Gamma^M: \quad 2^{2^{HB}} \to 2^{2^{HB}} \qquad \Gamma^M(I) = \min(\Gamma^{INT}(I))$

$\Gamma^{INT}: \quad 2^{HB} \to 2^{HB} \qquad \Gamma^{INT}(I) = \bigcup_{I \in I} MOD(\Gamma(I))$

where *MOD* gives all models of a state and min filters out the minimal models. The latter operator looks harmless; however this is a rather costly step. Its definition is given by: $\min(I) = \{I \in I \mid \neg\exists J \in I : J \subset I\}$. In [BFN96] we gave a proof that the branches of a hyper tableau correspond to partial models of the program and in particular that in fair derivations branches correspond to models. In [Nie96] it is demonstrated how the computation of the min-operator based on this definition can be avoided.

In the following we additionally depict the relation between one step with the Γ^M operator and hyper extension.

Definition 3.5
Let T be a hyper tableau and b an open branch. A *complete extension of T at b wrt. a set of program clauses $\mathcal{S}^+$* is a tree T' which can be obtained from T by applying as long as possible[4] hyper extension steps with clauses from $\mathcal{S}^+$ such that (a) only branches b' are selected which contain b as a prefix, and (b) in the hyper condition only literals from b are used, and (c) no extension step introduces an irregular branch. □

The following lemma establishes the connection of partial branches, i.e. models from a hyper tableau to the iterations using the Γ^Moperator.

Lemma 3.6
Let T be a hyper tableau consisting of one single branch b and let T' be a complete extension of T at b wrt. a set of program clauses $\mathcal{S}^+$. Then $\{lit(b') \mid b' \in T'\} \subseteq MOD(\Gamma(lit(b)))$

Note that since b is a single branch of a hyper tableau for S^+ each literal from $\mathrm{lit}(b)$ is contained in S^+ as a positive unit clause.

4 Hyper Tableaux and Restart Model Elimination

Unlike hyper tableaux, the RME calculus is a goal oriented interpreter for positive disjunctive logic programs [BF94a, BFS95]. It is a very simple extension of model elimination, which allows a procedural reading of disjunctive clauses. This is possible, because the calculus does not need any contrapositives. For a discussion of

[4] Obviously, as $\mathcal{S}^+$ can be infinite, this derivation is possibly infinite. In this case we take the chain limit of a branch to define the interpretation assigned to it; see [BFN96] for details.

these aspects the reader is referred to the above cited literature. Here we are interested only in the relation between RME to hyper tableaux, and therefore we only present its simplest variant.

RME is implemented by using the PTTP technique and hence it offers the advantage that in case of Horn programs the procedure *is the underlying PROLOG* system. Only the non-Horn part of a disjunctive logic program is treated by a compiler such that at run time a PROLOG program is executed by an efficient PROLOG system (for details see [BF94b]).

RME is a *top-town* calculus, i.e. derivations start with a (negative) goal clause and end at the (positive) facts. Our main result below shows how any closed hyper tableau can be transformed into a RME refutation. This transformation will essentially "reverse" a hyper tableau from the leaves to the root, where a splitting in hyper tableaux corresponds to a "restart step" in RME.

This result is in close relationship to the standard result in [Llo87] saying that any finite iteration of the T-operator over *definite* programs can be simulated top-town in a SLD-refutation. In fact, we generalize this result to the non-Horn case.

4.1 Restart Model Elimination

We will briefly review the RME calculus as presented in [BF94a]. However, for ease of presentation we will use a slightly different notation based on tableaux (Section 1) and following the style of Definition 2.1.

Definition 4.1 (Restart Model Elimination)
Let $\mathcal{S}$ be a finite, but not necessarily ground, clause set. We assume that $\mathcal{S}$ can be partitioned in[5] $\mathcal{S} = P \dot\cup \{ \leftarrow Q\}$, where the *query* $\leftarrow Q$ is a purely negative clause, i.e. it is of the form $\leftarrow B_1, \ldots, B_n$, and P is satisfiable.

Restart model elimination tableaux (RME tableaux) with substitution σ for $\mathcal{S}$ are inductively defined as follows:

Initialization step: A clausal tableau obtained by extending the root node of the empty tree by the query $\leftarrow Q \in \mathcal{S}$ is a hyper tableau for $\mathcal{S}$ with substitution $\sigma = \varepsilon$ (the empty substitution). In this context $\leftarrow Q$ is also called the *goal clause* of the tableau. All branches are labeled as "open".

Linked extension step: If

1. T is an open RME tableau for $\mathcal{S}$ with substitution σ_T, $f(T) = b$ (i.e. b is selected in T by f) with negative open leaf node $\neg A$, and
2. $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is a new variant of a clause from $\mathcal{S}$ ($m \geq 1$, $n \geq 0$), called *extending clause* in this context, and
3. σ is a most general unifier for A and some A_i (where $1 \leq i \leq m$),

then the literal tree $T'\sigma$ is a RME tableau for $\mathcal{S}$ with substitution $\sigma_T\sigma$, where T' is obtained from T by extending b by C, and then labeling the new branches

$$(b, A_1), \ldots, (b, A_{i-1}), (b, A_{i+1}), \ldots, (b, A_m), \ldots, (b, \neg B_1), \ldots, (b, \neg B_n)$$

[5] "$\dot\cup$" denotes disjoint union.

as "open", and labeling the new branch (b, A_i) as "closed".

Reduction step: If

1. T is an open RME tableau for $\mathcal{S}$ with substitution σ_T, $f(T) = b$ with negative open leaf node $\neg A$, and
2. $A' \in b$ is a positive literal in b, and
3. σ is a most general unifier for A and A',

then the literal tree $T'\sigma$ is a RME tableau for $\mathcal{S}$ with substitution $\sigma_T\sigma$, where T' is obtained from T by labeling b as "closed"[6]

Restart step: If

1. T is an open RME tableau for $\mathcal{S}$ with substitution σ, $f(T) = b$ (i.e. b is selected in T by f) with positive open leaf node A, and
2. $C = \leftarrow B_1, \ldots, B_n$ is a new variant of some negative clause from $\mathcal{S}$,

then the literal tree T' is a RME tableau for $\mathcal{S}$ with substitution σ_T, where T' is obtained from T by extending b by C.

The notions of *derivation* and *refutation* are taken from Definition 2.2. □

As an example consider the clause set in Figure 1 again. Figure 2 contains a RME refutation.

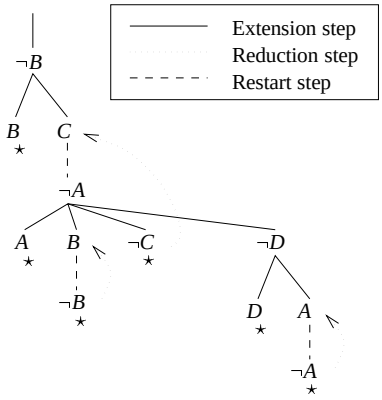


Figure 2: A RME refutation of the clause set of Example 5.

In [BFS95] we investigated the computation of answers by means of variants of RME. For the present paper we only restate one answer completeness result. For this we need the notion of an answer: if $\leftarrow Q$ is a query, and $\theta_1, \ldots, \theta_m$ are substitutions for the variables from Q, then $Q\theta_1 \vee \ldots \vee Q\theta_m$ is an *answer* (for P) An answer $Q\theta_1 \vee \ldots \vee Q\theta_m$ is a *correct answer* if $P \models \forall(Q\theta_1 \vee \ldots \vee Q\theta_m)$. Now let a RME refutation of S with goal clause $\leftarrow Q$ and substitution σ be given. Assume that this refutation contains m occurrences of the query, i.e. it contains one initialization step and $m-1$ restart steps with the clause $\leftarrow Q\rho_i$, where ρ_i is the renaming substitution of this step (ρ_i is the empty substitution for the initialization step). Let $\sigma_i = \rho_i\sigma|_{dom(\rho_i)}$. Then $Q\sigma_1 \vee \ldots \vee Q\sigma_m$ is a *computed answer* (for P).

That is, we simply collect applications of the instantiated query clause to obtain the answer. This idea is, of course, not new. For resolution, question answering was invented in the early paper [Gre69]; the idea is to attach answer literals to trace the usages of the query in the resolution proof (see also [CL73].

[6] Here, reduction steps are applied from negative leaf literals to positive ancestor literals; it would also be sound (but not neccessary for completeness) to allow reduction steps from positive leaf literals to negative ancestor literals. See [BF94a].

Theorem 4.2 (Answer-completeness of RME)
Let $\mathcal{S}$, P and $\leftarrow Q$ as in Definition 4.1, and let f be a selection function; let $Q\theta_1 \vee \ldots \vee Q\theta_l$ be a correct answer for P. Then there exists a RME refutation of $\mathcal{S}$ with computed answer $Q\sigma_1 \vee \ldots \vee Q\sigma_m$ such that $Q\sigma_1 \vee \ldots \vee Q\sigma_m$ entails $Q\theta_1 \vee \ldots \vee Q\theta_l$, i.e.

$$\exists\delta\,\forall i \in \{1,\ldots,m\}\,\exists j \in \{1,\ldots,l\}\; Q\sigma_i\delta = Q\theta_j.$$

Informally, the theorem states that for every given correct answer we can find a computed answer which can be instantiated by means of a *single* substitution δ to a subclause of the given answer, and hence implies it. To obtain this result we have to demand *one single* substitution δ which maps any of the instantiated query clauses $\leftarrow Q\rho_i\sigma$ used in extension steps to the respective clause on the ground level. Refinements and improvements of this result can be found in [BFS95].

4.2 Mapping Hyper Tableaux to Restart Model Elimination

As mentioned in the introduction to this section, our main result is a mapping from hyper tableaux to RME. Together with the results of the previous sections we thus have a top-down interpreter for the fixpoint semantics of positive disjunctive programs.

Theorem 4.3 (Top-Down Semantics for Hyper Tableaux)
Let T_H be a closed hyper tableau containing the tableau clauses $\mathcal{S}$[7]. Let $G = \leftarrow B_1,\ldots,B_n$ be some tableau clause in T_H (which hence closes a branch). Then there is a RME refutation of $\mathcal{S}$ with goal clause G.

Proof. Let $\mathcal{S}_H$ be the multiset of tableau clauses occurring in T_H. Let $k(\mathcal{S}_H)$ denote the number of occurrences of positive literals in $\mathcal{S}_H$ minus the number of non-negative clauses[8] in $\mathcal{S}_H$ ($k(\mathcal{S}_H)$ is a measure for the "Hornness" of $\mathcal{S}_H$; it is related to the well-known *excess literal parameter*). Now we prove the claim by induction on $k(\mathcal{S}_H)$.

Base case: $k(\mathcal{S}_H) = 0$. $\mathcal{S}_H$ and thus also $\mathcal{S}$ must be a set of Horn clauses. In this case the theorem rephrases in our setting the well-known corresponding result from [Llo87], which links the T-operator for definite programs to SLD-Resolution. A proof from scratch is in the full version of this paper.

Induction step: $k(\mathcal{S}_H) > 0$. As the induction hypothesis assume the result to hold for closed hyper tableau for clause sets $\mathcal{S}'_H$ satisfying $k(\mathcal{S}'_H) < k(\mathcal{S}_H)$. Figure 3 depicts the proof.

Some ancestor node A of the tableau clause $G = \leftarrow B_1,\ldots,B_n$ must have one or more positive brother nodes, because otherwise $\mathcal{S}_H$ would be a Horn multiset. Let $C = (A_1,\ldots,A_m, A \leftarrow \mathcal{B}) \in \mathcal{S}_H$ be the tableau clause where the node A is contained in. Here, $\mathcal{B}$ is understood as a (possibly empty) sequence of positive literals. Below we will also write expressions like $\neg\mathcal{B}$ and mean the clause $\bigvee_{B\in\mathcal{B}} \neg B$.

[7] The notion *tableau clause* is defined in the "Preliminaries" section

[8] A *non-negative clause* is a clause containing at least one positive literal.

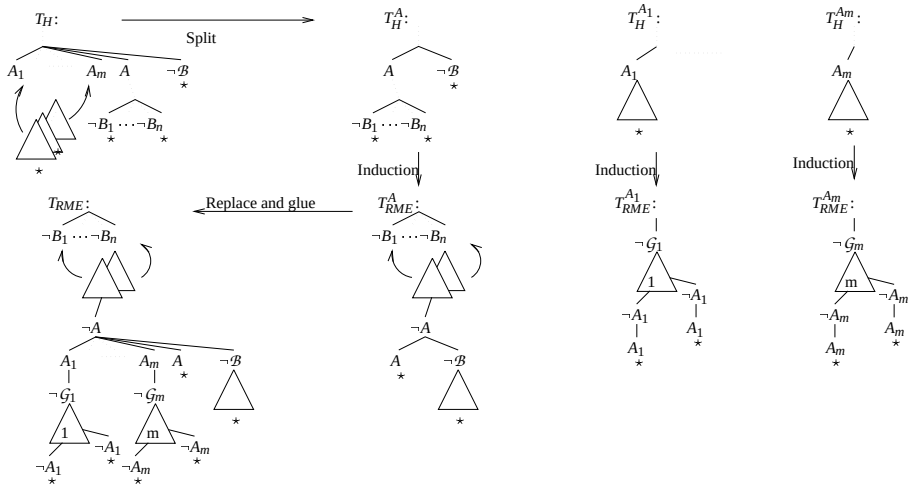


Figure 3: Proof of Theorem 4.3.

We split T_H into $m+1$ closed hyper tableaux: the hyper tableaux T_H^A is obtained from T_H by replacing the tableau clause C by $A \leftarrow \mathcal{B}$ (and thus deleting the subtrees below $A_1, \ldots, A_m$), and the hyper tableaux $T_H^{A_i}$ is obtained from T_H by replacing C by A_i (for $i = 1, \ldots, m$). The other parts of T_H are kept unchanged for for the splitted tableaux.

Let $\mathcal{S}_H^A$ and $\mathcal{S}_H^{A_i}$ be the tableau clause multisets corresponding to T_H^A and $T_H^{A_i}$. It holds that $k(\mathcal{S}_H^A) < k(\mathcal{S}_H)$ and $k(\mathcal{S}_H^{A_i}) < k(\mathcal{S}_H)$. Notice that T_H^A still contains G. Hence, by the induction hypothesis, there is a RME refutation T_{RME}^A[9] of $\mathcal{S}_H^A$ with goal clause G.

Similarly, by applying the induction hypotheses m times we learn that there are RME refutations $T_{RME}^{A_i}$ of $\mathcal{S}_H^{A_i}$ with some respective goal clauses $\leftarrow \mathcal{G}_i \in \mathcal{S}_H^{A_i}$. Since splitting does not affect the negative clauses it holds that $\leftarrow \mathcal{G}_i \in \mathcal{S}$. Hence, $T_{RME}^{A_i}$ is a RME refutation of $\mathcal{S} \cup \{A_i \leftarrow\}$. Notice that positive unit clauses like $A_i \leftarrow$ can be used in RME refutations only to close branches, without introducing new subgoals (as indicated in Figure 3).

Now we can put things together. Consider T_{RME}^A again. It (possibly) uses the clause $A \leftarrow \mathcal{B}$. However, this clause is (possibly) not contained in $\mathcal{S}$. In order to turn T_{RME}^A into a RME refutation of $\mathcal{S}$, we first replace every occurrence of the tableau clause $A \leftarrow \mathcal{B}$ in T_{RME}^A by C. This leaves us with open branches ending in (possibly several occurrences of) $A_1, \ldots, A_m$. Now, at each of these branches ending in A_i we can restart with the clause $\leftarrow \mathcal{G}_i$. Then we append below the upcoming tableau clause $\neg\mathcal{G}_i$ the refutation $T_{RME}^{A_i}$ *and* we replace possible extension steps in $T_{RME}^{A_i}$ with $A_i \leftarrow$ by reduction steps to the branch literal A_i where the restart occurred. As a result we get the desired RME refutation T_{RME} of $\mathcal{S}$ with goal clause G. Q.E.D.

[9] To be precise, there is a RME refutation having T_{RME}^A as its last element; but we will confuse this.

We consider the result of this Section –Theorem 4.3– as an *initial* investigation in the relationship between hyper tableaux and RME. It would be interesting to investigate the complexity of this mapping and to improve it. Currently each *single* hyper extension step might result in *many* extension and restart steps. It might be possible to improve the situation by additional RME inference rules like factoring.

5 SLO-Resolution

In [Raj89] SLO-Resolution is introduced as a generalization of SLD-Resolution. This interesting approach offers a goal-directed approach for the interpretation of positive disjunctive programs. In a subsequent paper, Rajesakar and Yusuf offer a modification of the WAM for an implementation.

At a first glance, SLO Resolution seems to be an alternative to the approach to disjunctive logic programming as we offer it. It is goal directed and it very close related to the state semantics. In fact the completeness proof is very much as the one from SLD-resolution, only the fixpoint semantics is different.

However there are two drawbacks which are fixed by our approach:

- When restricted to Horn clauses, the calculus is not equivalent to SLD-resolution, and
- there is only a ground completeness result; it is clear that SLO-resolution can not answer the query $\leftarrow p(x)$ with respect to the program $p(a), p(b) \leftarrow$.

RME is an extension of SLD-resolution and we have answer completeness of various variants. In this section we will demonstrate that SLO-resolution is very close related to bottom-up hyper tableaux. We show how to simulate SLO-resolution, by simply inverting the signs of all literals and then apply hyper tableaux. We do not claim that this transformation is original, it has been used e.g. in [Yah96] to turn a bottom-up prover into a goal-directed top-down one; moreover we want to point out that this simple technique can be used to simulate and to extend SLO-resolution.

The following definitions are taken from [Raj89].

A *goal* for a disjunctive program is of the form $\leftarrow (C_1, \ldots, C_n)$, where $n \geq 0$ and the C_i are positive clauses.

Definition 5.1
Let P be a positive disjunctive logic program and let G be a goal. An SLO-derivation from P with goal G consists of a (finite or infinite) sequence of goals $G_0 = G, G_1, ..,$ such that for all $i \geq 0$, G_{i+1} is obtained from $G_i = \leftarrow (C_1, \ldots, C_m, \ldots, C_k)$ as follows:

1. C_m is a clause in G_i. C_m is called the selected clause.
2. $C \leftarrow B_1, \ldots, B_q$ is a program clause in P.
3. C subsumes C_m with most general unifier θ.

4. G_{i+1} is the goal $\leftarrow (C_1,\ldots,C_{m-1},B_1\vee C_m,\ldots,B_q\vee C_m,C_{m+1},\ldots,C_k)\theta$

As usual derivations of the empty clause from G using P are called refutations; one also says that the goal G succeeds for P.

□

In [Raj89], Rajesakar gives a ground completeness result by induction over the fixpoint operator Γ on states. Without loss of generality we assume in the following only goals of the form $\leftarrow C$ where C is a positive disjunction. Note that a negative clause $\leftarrow A_1,\ldots,A_n$ is different from a goal $\leftarrow A_1\vee\ldots\vee A_n$; the latter is standing for a set of negative units.

Definition 5.2
The dual P^d of a clause $P = A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$ is obtained by inverting the arrow, i.e. $P^d = B_1,\ldots,B_m \leftarrow A_1,\ldots,A_n$. This could be alternatively formulated, by saying that signs of every literal in $P = A_1\vee\ldots\vee A_n\vee\neg B_1\vee\ldots\vee\neg B_m$ are complemented to get $P^d = \neg A_1\vee\ldots\vee\neg A_n\vee B_1\vee\ldots\vee B_m$.

Note that the dual of a goal $G = \leftarrow A_1\vee\ldots\vee A_n$ is the set of clauses $\{A_1\leftarrow, \ldots, A_n\leftarrow\}$, since G , written in clause form is the set of negative units $\{\leftarrow A_1, \ldots, \leftarrow A_n\}$. This transformation is extended to set of clauses in an obvious way. □

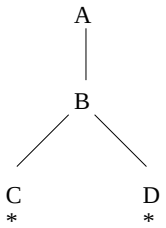
It is very easy to see that this transformation leaves unsatisfiability invariant. The following is a SLO-derivation of P with goal G from Example 1.

$$\leftarrow A\vee B \tag{5}$$
$$\leftarrow C\vee A\vee B,\quad D\vee A\vee B \qquad \text{from 5) and 1)} \tag{6}$$
$$\leftarrow D\vee A\vee B \qquad \text{from 6) and 2)} \tag{7}$$
$$\leftarrow \qquad \text{from 7) and 3)} \tag{8}$$

In order to use the hyper tableaux calculus to simulate this derivation we construct the dual program P^d and goal G^d.

$$\begin{array}{lll} P^d\colon & C,D\leftarrow A,B & \\ & \leftarrow B,C & \\ & \leftarrow A,D & \\ G^d\colon & A\leftarrow & B\leftarrow \end{array}$$

Applying hyper tableaux to this clause set gives a closed tableau depicted on the right. The starting goal 5 in the SLO-refutation corresponds to the fist two extension steps with the two facts $A\leftarrow$ and $B\leftarrow$ from the dual goal G^d, resulting in the tableau with the two nodes A and B. The SLO-step yielding in line 6 the goal $\leftarrow C\vee A\vee B,\quad D\vee A\vee B$ corresponds to a hyper extension step with $C,D\leftarrow A,B$. The two branches from the tableau in the



right figure are coded in line 6 by the two clauses in the goal. The step resulting in goal 7 corresponds to the extension step with $\leftarrow B, C$ and the last step to the extension with $\leftarrow A, D$.

Lemma 5.3
Given a ground SLO-derivation from P with ground goal $\leftarrow C$ and derived goal $\leftarrow C_1, \ldots, C_m$. Then there is a hyper tableau T for P^d and a substitution σ such that for all $b \in T$ there is a C_i containing $L\sigma$, for any label L from b.

Based on the previous lemma we are currently investigating how SLO-resolution can be improved by applying the concepts of hyper tableaux. By this it is possible to make SLO-resolution complete with respect to logical consequences and to get rid of some of the rigidly treated variables.

6 Conclusion

We investigated the relation between the top-down restart model elimination (RME) and the bottom-up hyper tableaux calculus. As our main results, we demonstrated that this hyper tableaux calculus can be seen as a fixpoint semantics for DLPs and that restart model elimination provides a corresponding top-down proof procedure.

We want to point out that hyper tableaux can be used as well for efficient model generation. This is of particular interest, when non-monotonic extensions for DLP have to be implemented. As a base for this, we are currently investigating two kinds of minimal reasoning. One uses the hyper tableaux calculus for computing minimal models: in [Nie96] it is shown how minimal models can be computed *without* keeping and comparing models in memory, by means of hyper tableaux. The other approach from [Ara96] uses RME as a base calculus to compute the generalized closed world assumption.

Future work will be the incorporation of negation into DLPs and to investigate more closely the relation between hyper tableaux and RME.

References

[Ara96] C. Aravindan. An abductive framework for negation in disjunctive logic programming. In *Proc. JELIA 96*, number 1126 in LNAI. European Workshop on Logic in AI, Springer, 1996.

[BF94a] P. Baumgartner and U. Furbach. Model Elimination without Contrapositives and its Application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994. Short version in: Proceedings of CADE-12, Springer LNAI 814, 1994, pp 87–101.

[BF94b] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *LNAI*, pages 769–773. Springer, 1994. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[BFN96] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in LNAI. European Workshop on Logic in AI, Springer, 1996.

[BFS95] P. Baumgartner, U. Furbach, and F. Stolzenburg. Model Elimination, Logic Programming and Computing Answers. In *14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, volume 1, 1995.

[CL73] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[Dec91] H. Decker. A model-theoretic and a fixpoint-semantic foundation of first order databases. Siemens AG, München, 1991.

[FM91] J.A. Fernandez and J. Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In Koichi Furukawa, editor, *Proc. 8th International Conference on Logic Programming*, pages 660–675, 91.

[Fur92] U. Furbach. Computing answers for disjunctive logic programs. In Pearce and Wagner, editors, *Logics in AI, JELIA'92*. Springer, LNAI 633, 1992.

[Gre69] C. Cordell Green. Theorem-Proving by Resolution as a basis for Question-Answering Systems. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier Publishing Company, Inc., 1969.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer, second, extended edition, 1987.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.

[Lov78] D.W. Loveland. *Automated Theorem Proving - A Logical Basis*. North Holland, 1978.

[Lov87] D.W. Loveland. Near-Horn Prolog. In J.-L. Lassez, editor, *Proc. of the 4th Int. Conf. on Logic Programming*, pages 456–469. The MIT Press, 1987.

[MB88] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. 9th CADE*. Argonnee, Illinois, Springer LNCS, 1988.

[MR90] J. Minker and A. Rajasekar. A fixpoint semantics for disjunctive logic programs. *J. Logic Programming*, 9:45–74, 1990.

[Nie96] I. Niemelä. A Tableau Calculus for Minimal Model Reasoning. In P. Moscato, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, number 1071 in Lecture Notes in Artificial Intelligence. Springer, 1996.

[Pla88] D. Plaisted. Non-Horn Clause Logic Programming Without Contrapositives. *Journal of Automated Reasoning*, 4:287–325, 1988.

[Raj89] A. Rajasekar. *Semantics for Disjunctive Logic Programs*. PhD thesis, University of Maryland, 1989.

[RS90] D.W. Reed and B.T. Smith. A case-analysis based fixpoint semantics for disjunctive propgrams. Extended abstract for the Workshop on 'Non-Hornclause programming' during NACLP'90, 1990.

[SMR95] D. Seipel, J. Minker, and C. Ruiz. Model generation and state generation for disjunctive logic programs. Technical report, Univ. of Tübingen, 1995.

[Sti90] M. Stickel. A Prolog Technology Theorem Prover. In M.E. Stickel, editor, *Proc CADE 10, LNCS 449*, pages 673–675. Springer, 1990.

[Yah96] A. Yahya. Query Answering in Disjunctive Deductive Databases. Dagstuhl-Seminar on *Disjunctive logic programming and databases: Non-monotonic aspects*, 1996.