Hyper Tableaux

Peter Baumgartner · Ulrich Furbach · Ilkka Niemelä

Universität Koblenz
Institut für Informatik
Rheinau 1, 56075 Koblenz, Germany
E-mail: {peter,uli,ini}@informatik.uni-koblenz.de

Abstract. This paper introduces a variant of clausal normal form tableaux that we call "hyper tableaux". Hyper tableaux keep many desirable features of analytic tableaux while taking advantage of the central idea from (positive) hyper resolution, namely to resolve away all negative literals of a clause in a single inference step. Another feature of the proposed calculus is the extensive use of universally quantified variables. This enables new efficient forward-chaining proof procedures for full first order theories as variants of tableaux calculi.

1 Introduction

This paper introduces a variant of clausal normal form tableaux that we call "hyper tableaux". Hyper tableaux keep many desirable features of analytic tableaux while taking advantage of the central idea from (positive) hyper resolution. From tableau calculi we benefit from the following features:

– Tableau calculi offer a rich structure for the whole derivation process; important parts of the derivation history are stored in a tableau and can be used for subsequent optimizations.
– As a byproduct of this structure we get, in our case, a model construction procedure. At any state within a derivation each branch of the tableau is a representation of a partial model for the given set of clauses. This property enables efficient minimal model reasoning, as demonstrated in [Niemelä, 1996b, Niemelä, 1996a].
– For disjunctive datalog, i.e. first order logic without function symbols our calculus offers a decision procedure, by applying the subsumption technique introduced.

From resolution we incorporate the following:

– The "hyper-property", namely to resolve away all negative literals of a clause in a single inference step. From a resolution standpoint our calculus can be described as a positive hyperresolution calculus plus additional structure for controlling the generation of new clauses. In saturation based theorem proving the latter turned out to be a major problem; our calculus offers a solution by using a tableau structure and thus enabling refinements like *optional* factorization, regularity, level cut and efficient memory management. In resolution terminology, the "level cut" facility enables a more goal-oriented behavior of hyper-resolution by deletion of irrelevant derived clauses; by "memory management" we mean the possibility to delete derived clauses by simple looking at the tableaux structure instead of using a subsumption-based search through the clauses.

- We further make extensive use of universally quantified variables. These enables "subsumption" as the primary pruning technique, which is much stronger than the usual clausal tableaux pruning technique based on syntactically equal literals. This results in new efficient forward-chaining proof procedures for full first order theories as variants of tableaux calculi.

There is previous work on defining forward-chaining provers like SATCHMO ([Manthey and Bry, 1988, Loveland *et al.*, 1995]) and the MGTP system ([Fujita and Hasegawa, 1991]). These systems are high-performance theorem provers and a parallel version of MGTP even discovered new mathematical results in finite algebra. However, these approaches have the disadvantage that they are only applicable to range restricted clauses. Of course, there is a work-around which is used in these provers, namely the introduction of so called domain predicates. This results in an enumeration of Herbrand terms, thus giving up the full power of unification. Our experiments demonstrate that there are a lot of examples where the domain enumeration is a real disadvantage.

Our work can be seen as a formalization and extension of work done around SATCHMO. It extends the above cited work done in the following sense:

- We do not restrict to range restricted clauses.
- In SATCHMO the "domain predicate" technique is employed, which guarantees that the tableau under construction contains *ground* literals only. We use the domain predicate technique in a much more restricted way, namely for those clauses only which contain a common variable in more than one positive literal. Since this can, of course, never be the case for Horn clauses, we get as consequence positive hyper resolution for Horn clauses. Where SATCHMO would put ground instances of the unit clause, say, $P(x)$ on the branch, we put the more compact formula $\forall x\, P(x)$ there instead.
- We give a rigorous completeness proof.

Because of the implicitly universally quantification the semantics of our variables is different from the standard semantics for "free" tableaux variables [Fitting, 1990], according to which a free variable is a placeholder for a *single* term (also called a *rigid* variable). One crucial point is that our "universal" variables allow to employ "subsumption" instead of the weaker, usual test for identical branch literals in tableaux with rigid variables. Further, unlike as in rigid variable tableaux, our extension inference rule does not act on the variables of the tableaux to be extended. We can thus look at "one branch at a time", isolated from the other branches and the variables there.

Another important point is that it seems extremely difficult to define a fairness condition for derivations of tableaux with rigid variables which can be reasonably implemented. By "reasonably implementable" we exclude the widely used backtracking oriented iterative deepening over increasing resource bounds. Such a scheme, as it is proposed e.g. in [Fitting, 1990, Beckert and Hähnle, 1992] (the latter deals with a mixed rigid/universal variable model) is in principle not necessary for proof confluent calculi like analytic tableau.

Our hyper tableaux calculus is proof confluence and hence it can be implemented in a straightforward way without backtracking over the generated tableaux.

In [Hähnle, 1995] a tableaux calculus is presented and proven to be complete, which is very similar to the ground version of hyper tableaux. However, it is left open there how to lift the proof to the first order level (which is far from being trivial). Also implementational issues are not considered. Our paper can thus also be seen as a contribution to solve open issues in that paper.

The rest of this paper is structured as follows: after defining the formal framework in the next section we introduce in Section 3 the hyper tableaux calculus. In Section 4, the main part of the paper, a strongly complete variant of the calculus for the first-order case is introduced together with a powerful redundancy criterion and a fairness condition. Finally we discuss a proof procedure and report on practical experiments.

2 Preliminaries

In what follows, we assume that the reader is familiar with the basic concepts of first-order logic. A *clause* is a multiset of literals, usually written as the disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ or the implication $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ ($m \geq 0, n \geq 0$). As usual, the variables occurring in clauses are considered implicitly as being universally quantified, a clause is considered logically as a disjunction of literals, and a (finite) clause set is taken as a conjunction of clauses. A ground clause is a clause containing no variables. Literal K is an *instance* of literal L, written as $K \geq L$ or $L \leq K$, iff $K = L\gamma$ for some substitution γ. Let $\overline{L}$ denote the complement of a literal L. Two literals L and K are *complementary* if $\overline{L} = K$.

Let X be a literal or a clause. X^g is the set of all ground instances of X (wrt. a given signature which contains at least one constant symbol). Similarly, if $\mathcal{X}$ is a clause set or literal set, then $\mathcal{X}^g := \bigcup_{X \in \mathcal{X}} X^g$.

Definition 1 (Literal tree, Clausal Tableau) [Letz *et al.*, 1994]. A *literal tree* is a pair (t, λ) consisting of a finite, ordered tree t and a labeling function λ that assigns a literal to every non-root node of t. The *successor sequence* of a node N in an ordered tree t is the sequence of nodes with immediate predecessor N, in the order given by t.

A *(clausal) tableau* T of a set of clauses $\mathcal{S}$ is a literal tree (t, λ) in which, for every successor sequence $N_1, \ldots, N_n$ in t labeled with literals $K_1, \ldots, K_n$, respectively, there is a substitution σ and a clause $\{L_1, \ldots, L_n\} \in \mathcal{S}$ with $K_i = L_i\sigma$ for every $1 \leq i \leq n$. $\{K_1, \ldots, K_n\}$ is called a *tableau clause* and the elements of a tableau clause are called *tableau literals*.

Definition 2 (Branch, Open and Closed Tableau, Selection Function). 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. We say branch $b = N_0, \ldots, N_n$ is a *prefix* of branch c, written as $b \leq c$ or $c \geq b$, iff $c = N_0, \ldots, N_n, N_{n+1}, \ldots, N_{n+k}$ for some nodes $N_{n+1}, \ldots, N_{n+k}$, $k \geq 0$.

The *branch literals* of branch $b = N_0, \ldots, N_n$ are the set $\mathrm{lit}(b) = \{\lambda(N_1), \ldots \lambda(N_n)\}$. 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 $A \in \mathrm{lit}(b)$.

In order to memorize the fact that a branch contains a contradiction, we allow to label a branch as either *open* or *closed*. A tableau is *closed* if each of its branches is closed, otherwise it is *open*.

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*.

Note that branches are always finite, as tableaux are finite.

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.

Definition 3 (Branch Semantics). Let $\mathcal{L}$ be a possibly infinite set of literals. Define $\mathcal{L}^{\forall} := \{\forall L | L \in \mathcal{L}\}$ as the *clause set* of $\mathcal{L}$, where $\forall F$ denotes the universal closure of formula F. Whenever we take an atom set $\mathcal{A}$ where a set of formulae were required, we implicitly assume its clause set $\mathcal{A}^{\forall}$. By the *model of an atom set $\mathcal{A}$* we mean the minimal Herbrand model of $\mathcal{A}^{\forall}$ which we denote by $[\![\mathcal{A}]\!]$. Using a previous convention, we thus identify in particular a branch b with the clause set $(\text{lit}(b))^{\forall}$. Hence, it is meaningful to say that a branch b is unsatisfiable, and also $[\![b]\!] \models C$ is defined (the least Herbrand model of the clause set of b satisfies the clause C).

Based on the above definitions we now introduce hyper tableaux and the inference steps of the calculus.

3 The Calculus

We are going to define the calculus of hyper tableaux as a process which generates a restricted form of clausal tableaux. For this, we need one more preliminary definition.

Definition 4 (Pure clause). A clause $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is called *pure* iff variables are not spread over distinct head literals, i.e. iff $\text{Var}(A_i) \cap \text{Var}(A_j) = \emptyset$, for $i, j \in \{1, \ldots, m\}$ and $i \neq j$. A substitution π is a *purifying substitution* for C iff $C\pi$ is pure.

Obviously, every non-pure clause can be turned into a pure instance thereof by application of an appropriate substitution.

Definition 5 (Hyper tableau). Let S be a finite set of clauses and f be a selection function. *Hyper tableaux* for S are inductively defined as follows:
Initialization step: A one node literal tree is a hyper tableau for S. Its single branch is marked as "open".
Hyper extension step: If

1. T is an open hyper tableau for S, $f(T) = b$ (i.e. b is selected in T by f) with open leaf node N, and
2. $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is a clause from S ($m \geq 0$, $n \geq 0$), called *extending clause* in this context, and

3. σ is a most general substitution[1] such that $[\![b]\!] \models \forall(B_1 \wedge \cdots \wedge B_n)\sigma$ (referred to as *hyper condition*), and
4. π is a purifying substitution for $C\sigma$,

then the literal tree T' is a hyper tableau for S, where T' is obtained from T by attaching $m+n$ child nodes $M_1,\ldots,M_m,N_1,\ldots,N_n$ to b with respective labels

$$A_1\sigma\pi,\ldots,A_m\sigma\pi,\neg B_1\sigma\pi,\ldots,\neg B_n\sigma\pi$$

and marking every new branch $(b,M_1),\ldots,(b,M_m)$ with positive leaf as "open", and marking every new branch $(b,N_1),\ldots,(b,N_n)$with negative leaf as "closed".

We will write the fact that T' can be obtained from T by a hyper extension in the way defined as $T \vdash_{b,C,\sigma,\pi} T'$, 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.

Note that we do *not* take new variants, and that the substitution $\sigma\pi$ is *not* applied to the whole tableau but only to the extending clause. Condition 3, the *hyper condition*, expresses that *all* (instantiated) body literals have to be satisfied by the branch to be extended. This similarity to hyper *resolution* [Robinson, 1965] coined the name "hyper tableaux".

Expressing the hyper condition slightly different, we mark a branch as "closed" if and only if it is unsatisfiable. For instance, a branch containing literals $P(x)$ and $\neg P(y)$ is closed. In the standard tableaux with rigid variables (e.g. in [Fitting, 1990]) a branch is considered as closed if it contains a complementary pair of literals (notice that $P(x)$ and $\neg P(y)$ are not complementary). Of course, these notions coincide in the ground case.

The need for a purifying substitution in condition 4 in hyper extension step will guarantee the soundness of hyper tableaux calculi. The underlying property is the easy to prove observation that $\forall(A \vee B) \equiv (\forall A \vee \forall B)$ holds if clause $A \vee B$ is pure. The substitutions σ and π have to be applied in this order because if applied in exchanged order, there is no guarantee that the resulting instance of the extension clause is pure. This would destroy soundness.

Example 1. For illustration consider the single-literal branch $b = r(f(X))$ and the clause $C = p(X), q(X,Y) \leftarrow r(X)$. Then, $[\![b]\!] \models \forall r(X)\sigma$, where $\sigma = \{X \leftarrow f(X')\}$. The head $(p(X),q(X,Y))\sigma = p(f(X')), q(f(X'),Y)$ is impure. Taking e.g. a purifying substitution $\pi = \{X' \leftarrow a\}$ enables a hyper extension step, yielding the hyper tableau whose two open branches are $b_1 = (r(f(X)), p(f(a)))$ and $b_2 = (r(f(X)), q(f(a),Y))$. Now, the intended model candidates for the input clause set are just $[\![b_1]\!]$ or $[\![b_2]\!]$. It is important to note that the models are derived "locally" from the paths *alone*, but not from the whole tableaux. However, for this construction to be sound we have to require that $\forall b_1 \vee \forall b_2$ is a logical consequence of $\forall b$, which indeed holds due to the application of π.

[1] Here, "most general" means that whenever $[\![b]\!] \models \forall(B_1 \wedge \cdots \wedge B_n)\delta$ for some substitution δ, then $\sigma \leq \delta\ [\mathrm{Var}(B_1 \wedge \cdots \wedge B_n)]$. The notation $\sigma \leq \delta\ [V]$ means the restriction of the "more general" relation $\leq$ to the variables V. See [Siekmann, 1989].

We turn again back to Definition 5. The hyper condition in hyper extension step is — intentionally – given in a pure semantical way. With view to a proof procedure it is mandatory to decide (and not only to semi-decide) whether a clause C and most general substitution σ as required exist. Fortunately, this is possible:[2]

Proposition 6 (Implementing the Hyper Condition). *For every finite atom set $\mathcal{A}$ and conjunction of atoms $C = B_1 \wedge \cdots \wedge B_n$: if there is a substitution γ for C such that $[\![\mathcal{A}]\!] \models \forall(B_1 \wedge \cdots \wedge B_n)\gamma$ then there is a SLD resolution refutation of the clause set $P = \mathcal{A} \cup \{\neg B_1 \vee \ldots \vee \neg B_n\}$ with computed answer $\sigma \leq \gamma\,[Var(C)]$ and using exactly $|C|$ resolution steps. If there is no such γ, then each of the finitely many SLD derivations of P finitely fails.*

Notice that the input clause set for SLD resolution is very simple: it consists of only one negative clause and some positive unit clauses. We prefer this formulation over the unit hyper resolution procedure in [Chang and Lee, 1973] because its answer completeness result gives us immediately that σ is a *most general* substitution as required in the hyper condition.

The hyper extension step has the property that a branch is closed if and only if it ends in a negative literal. Thus it holds:

Proposition 7. *Every hyper tableau is a clausal tableau where every inner node is labeled with a positive literal. The converse does in general not hold.*

It is this property that motivates us to take the body literals of extending clauses into the extended tableaux; leaving them away would no longer give clausal tableaux (however, this is not a crucial point).

Definition 8 (Hyper Tableaux Derivation). Let $\mathcal{S}$ be a finite clause set, called the *set of input clauses*, and 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 $i > 1$, $T_{i-1} \vdash_{b_{i-1}, C_{i-1}, \sigma_{i-1}, \pi_{i-1}} T_i$ for some clause $C_{i-1} \in \mathcal{S}$, and some substitutions σ_{i-1} and π_{i-1}. This is also written as

$$T_1 \vdash_{b_1, C_1, \sigma_1, \pi_1} T_2 \cdots T_n \vdash_{b_n, C_n, \sigma_n, \pi_n} T_{n+1} \cdots$$

A hyper 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 tableau. Figure 1 shows an example refutation.

We comment on the relation to hyper resolution. Consider first the special case of Horn theories. Any hyper tableaux refutation develops a tableaux with *one single* open branch, which is closed in the concluding step. In this branch, the literals are either given positive unit clauses or positive literals derived by means of definite input clauses from the branch. Further, since all input clauses are Horn, they (and all their instances) are

[2] The missing proofs are contained in the long version. It can be obtained in the WWW using the URL `http://www.uni-koblenz.de/universitaet/fb4/publications/GelbeReihe/RR-8-96.ps.gz`

(1) $r(a), r(f(Z)) \leftarrow$
(2) $p(X), q(X,Y) \leftarrow r(X)$
(3) $p(X) \leftarrow q(X,Y)$
(4) $\leftarrow p(a)$
(5) $\leftarrow p(f(b))$

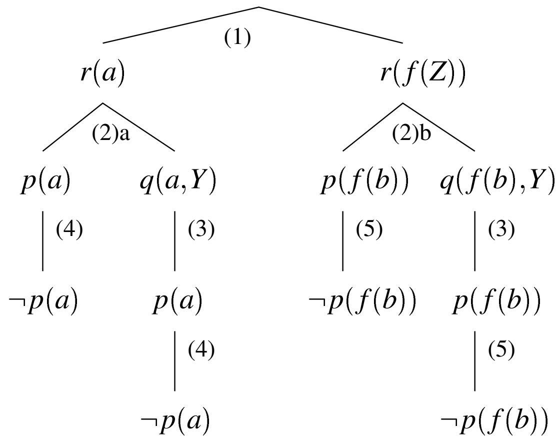

Fig. 1. A sample hyper tableaux refutation. The clause set is given by clauses (1)–(5). Variables are written in capital letters. The usage of the clauses in extension steps is indicated at the edges. The initial tableaux is set up with clause (1) (there is no other choice). Extension at $r(a)$ with clause (2) uses $\sigma = \{x \leftarrow a\}$; since Y is pure in the resulting head, we can choose $\pi = \varepsilon$ which leaves us Y as a universal variable. This extension step is indicated as "(2)a" (the body literals are not depicted). The further refutation of the left subtree should be obvious. For the right subtree we can extend $r(f(Z))$ with clause (2) again: first compute $\sigma = \{x \leftarrow f(Z)\}$. The resulting head $p(f(Z)) \vee q(f(Z),Y)$ of clause (2) is not pure; we guess $\pi = \{Z \leftarrow b\}$ in order to find the refutation immediately.

trivially pure. Hence there is never need for a purifying substitution. In this case "hyper tableaux" is the same as hyper*resolution* (with forward subsumption).

Now for the general case. Consider Example 1 again. Hyper resolution when applied to the clauses $r(f(X))$ (corresponding to the branch b) and clause C yields the clause $p(f(X)) \vee q(f(X),Y)$, which is, of course, different to the purified instance, e.g. $p(f(a)) \vee q(f(a),Y)$. Due to this purifying of variables it is possible to split the head of a rule as it is done by a hyper extension step. In hyper resolution terminology, this allows to use only the units from a branch as satellites of an hyper resolution step. In a hyper resolution calculus this can only be achieved by introducing an extra splitting rule.

Using the tableaux in Figure 1 again we can argue for one more interesting difference to hyper-resolution. There exists a hyper-resolution derivation of $r(f(Z))$ which is similar in structure to the closed hyper tableaux below the left branch in Figure 1. All the derived (positive) clauses in that derivation, such as $p(a) \vee q(a,Y) \vee r(f(Z))$, are subsumed now by $r(f(Z))$. However, in hyper tableaux no *search* through the clauses is necessary to delete the respective purified instances, because they all belong to the closed subtree below $r(a)$. This observation is what we meant by the term "memory management" in the introduction, and it serves as an example where taking a tableaux view can contribute to resolution.

4 Fairness, Redundancy and Completeness

We develop a completeness result of the above defined calculus using a model construction technique for open hyper tableau. The underlying *fairness* condition guarantees that *any* derivation eventually leads to a refutation (for unsatisfiable clause sets, of course). The result allows to include a redundancy criterion based on *subsumption* for search space pruning.

4.1 Redundancy, I-Paths and Fairness

For practical reasons, tableaux calculi should use a "regularity check" which forbids to have repetitions of the same literal along along a branch.

Definition 9 (Redundancy). A ground clause C is *redundant* in a (possibly infinite) set of atoms $\mathcal{A}$ iff $[\![\mathcal{A}]\!] \models C$; on the general level, a clause C is redundant in $\mathcal{A}$ iff $[\![\mathcal{A}]\!] \models C'$ for every ground instance $C' \in C^g$.

Intentionally, a clause is redundant iff each of its ground instances is satisfied by the interpretation given by $\mathcal{A}$. It is, of course, different to say that the universal closure of the clause is satisfied by the minimal interpretation given by $\mathcal{A}$.

Furthermore, it would be different to use the condition $\mathcal{A} \models C$ instead (i.e. every, not necessarily minimal, model of $\mathcal{A}$ is a model of C). The difference is important: consider e.g. the single literal branch B and the clause $A \leftarrow B, C$. It holds $[\![B]\!] \models A \leftarrow B, C$, however $B \not\models A \leftarrow B, C$. Thus, using the definition as it stands more clauses are redundant. More severely, in the alternate approach we were forced to extend branch B with $A \leftarrow B, C$ in order to satisfy it. However, this is obviously not possible by a hyper extension step.

Our notion of redundancy covers the above-mentioned regularity check, because if A is on a branch and hyper extension results in a second occurrence of A, then the tableaux clause of which A is part of, say $\ldots, A, \ldots \leftarrow B_1, \ldots, B_n$ is a logical consequence of A, and hence redundant. For hyper tableaux we can easily allow a stronger condition than regularity based on literal subsumption:

Lemma 10 (Sufficient Redundancy Criterion). *Let $\mathcal{A}$ be an atom set and $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ be a clause. If $[\![\mathcal{A}]\!] \models B_1 \wedge \cdots \wedge B_n$ implies $A' \leq A_i$, for some variant A' of $A \in \mathcal{A}$ and some $i \in \{1, \ldots, m\}$, then C is redundant in $\mathcal{A}$.*

For instance, in the example in Figure 1, the clause $r(f(g(X))), q(X) \leftarrow r(X)$ would be redundant in the atom set given by the branch up to point (2)b, because $r(f(X')) \leq r(f(g(X)))$, where $r(f(X'))$ is a variant of the branch literal $r(f(X))$.

In order to formalize fairness, we need one more preliminary concept ("i-paths"). For this, we always suppose a selection function as given which, however, will not referred to explicitly. Furthermore, $\mathcal{D}$ always refers to a derivation written as

$$\mathcal{D} = T_0 \vdash_{b_0, C_0, \sigma_0, \pi_0} T_1 \cdots T_n \vdash_{b_n, C_n, \sigma_n, \pi_n} T_{n+1} \cdots$$

Definition 11 (I-Path). Let b_k be a selected branch in $\mathcal{D}$. Then an *i-path* (infinite path) starting from b_k is a sequence $b_k (= b_{k+i_0}), b_{k+i_1}, b_{k+i_2}, \ldots$ of branches such that

1. $b_k (= b_{k+i_0}) \leq b_{k+i_1} \leq b_{k+i_2} \leq \cdots$
2. $i_j < i_{j+1}$ for all $j \geq 0$ (strictness) and
3. $\forall l \geq k \, \exists j \geq l$ such that b_{k+j} appears in the sequence (infiniteness).

Hence, i-paths are just sequences of branches that are infinitely often extended in the derivation. Note that for a finite derivation there are no i-paths.

To guarantee fairness it is sufficient to require that if some extension step is possible for a clause in a branch that is infinitely often extended, the clause becomes redundant at some point for each infinitely often extended continuation of the branch. To formalize this we need a notion of redundancy for an i-path.

Definition 12 (Path semantics, Redundancy in a Path). Where appropriate, we identify an i-path $p = \quad b_k(= b_{k+i_0}), b_{k+i_1}, b_{k+i_2}, \ldots$ with its atom set as follows: $\mathcal{A}(p) = \bigcup_{j \geq 0} \mathcal{A}(p)_j$, where $\mathcal{A}(p)_j = \mathrm{lit}(b_{k+i_j})$. We have to generalize Definition 9 towards paths: a clause $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ is *redundant in an i-path p* iff C is redundant in $\mathcal{A}(p)$.

Thus, given a path, we look at the atoms of its chain limit in order to determine redundancy.

Definition 13 (Fairness). The derivation $\mathcal{D}$ from a clause set $\mathcal{S}$ is called *fair* iff for all $k \geq 0$, $T_k \vdash_{b_k, C, \sigma, \pi} T'$ for some tableau T' implies that $C\sigma\pi$ is redundant in every i-path starting from b_k.

Notice that all finite derivations are fair and fairness is an issue only when infinite derivations are concerned. To state the completeness results we need a notion of a *finished derivation* by which we mean an infinite derivation or a derivation where either a closed tableau or a tableau with a *finished branch* is obtained. A branch is finished when we know that it cannot be closed no matter what extension steps are taken. The notion of redundancy can be employed to formalize this notion.

Definition 14 (Finished Branch). A branch in a tableau in a derivation from clause set $\mathcal{S}$ is called *finished* iff every clause in $\mathcal{S}$ is redundant in the branch.

A derivation from clause set $\mathcal{S}$ is called *finished* iff (i) there is closed tableau in the derivation or (ii) there is a tableau with a finished branch in the derivation or (iii) the derivation is infinite.

4.2 Completeness

There exist various completeness results which could be considered to be applicable to our case. The SATCHMORE program is proven complete in [Loveland *et al.*, 1995] for range-restricted programs (thus only ground tableau are considered). We are aware of the fairness-based completeness results for *ground* calculi of our type in [Hähnle, 1995, Fujita and Hasegawa, 1991]. Unfortunately, the widely used standard lifting proof technique (see e.g. [Fitting, 1990] for the tableau case), and also the refined approach with universal formulas of [Beckert and Hähnle, 1992], is not applicable in our case because it would only give us the *existence* of a hyper tableau refutation. Since we aim at a completeness result for *every* (fair) strategy, we have to develop a new proof from scratch.

Theorem 15 (Models of Open Hyper Tableaux.). *Let $\mathcal{D}$ be a (possibly infinite) fair finished derivation* $\mathcal{D} = T_0 \vdash_{b_0,C_0,\sigma_0,\pi_0} T_1 \cdots T_n \vdash_{b_n,C_n,\sigma_n,\pi_n} T_{n+1} \cdots$ *from a possibly infinite, possibly non-ground, clause set $\mathcal{S}$, such that every T_i ($i \geq 0$) is open. Then $\mathcal{S}$ is satisfiable.*

Proof. If the derivation is finite, there is a tableau with a finished open branch b. Hence, by Definition 14, every clause $C \in \mathcal{S}$ is redundant in b. By definition of redundancy (Def. 9) this is the same as $[\![b]\!]C$. In other words, $\mathcal{S}$ is satisfiable by virtue of $[\![b]\!]$.

Otherwise the derivation is infinite and there is an i-path $p = b_0, b_{0+i_1}, b_{0+i_2}, \ldots$ starting from b_0. We show $[\![\mathcal{A}(p)]\!] \models \mathcal{S}$. Since we deal with Herbrand interpretations this is equivalent to $[\![\mathcal{A}(p)]\!] \models \mathcal{S}^g$. Now, suppose, to the contrary, that $[\![\mathcal{A}(p)]\!] \not\models \mathcal{S}^g$ holds. Hence

$$[\![\mathcal{A}(p)]\!] \not\models C' \qquad \text{for some ground clause } C' \in \mathcal{S}^g. \tag{1}$$

The clause C' is of the form $C' = A'_1, \ldots, A'_m \leftarrow B'_1, \ldots, B'_n$ for some corresponding clause $C = A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ from $\mathcal{S}$. Now, Equation 1 implies

$$[\![\mathcal{A}(p)]\!] \models B'_1 \wedge \cdots \wedge B'_n \quad \text{and} \quad [\![\mathcal{A}(p)]\!] \not\models A'_1 \vee \cdots \vee A'_m \tag{2}$$

From 2 we conclude that there exists a *finite* subset $\mathcal{A}' \subseteq \mathcal{A}(p)$ such that $[\![\mathcal{A}']\!] \models B'_1 \wedge \cdots \wedge B'_n$. Recall that $\mathcal{A}(p)$ is the chain limit of every increasing atom sets $\mathcal{A}(p)_0 \subseteq \mathcal{A}(p)_1 \subseteq \cdots$. Hence, $\mathcal{A}' \subseteq \mathcal{A}(p)_l$, for some l, where $\mathcal{A}(p)_l = \mathrm{lit}(b_{0+i_l})$. Now we know $[\![b_{i_l}]\!] \models B'_1 \wedge \cdots \wedge B'_n$. By virtue of C', a hyper extension step $T_{i_l} \vdash_{b_{i_l},C,\sigma,\pi} T'$ exists, where σ and π are appropriate substitutions such that C' is a ground instance of $C\sigma\pi$. By fairness, $C\sigma\pi$ is redundant in every i-path starting from b_{i_l}. Hence, $C\sigma\pi$ is in particular redundant in the i-path[3] $p' = b_{i_l}, b_{i_{l+1}}, \ldots$, where $p = (b_0, b_{i_1}, \ldots, b_{i_{l-1}}) \circ p'$. Thus, since p and p' are the same wrt. limits, $C\sigma\pi$ is trivially also redundant in p. But then, by the definition of redundancy, C' is redundant in p, too. This means just $[\![\mathcal{A}(p)]\!] \models C'$ which plainly contradicts the choice of C' (Equation 1). Hence, the assumption must be wrong, and thus the theorem follows.

For theorem proving applications the converse direction of the previous theorem usually is more interesting: from a given (possible infinite) set of *unsatisfiable* clauses infer that a refutation exists, i.e. that a tableau is derivable where every branch is closed.

It is clear that once a closed tableau is derived, the derivation cannot be continued, because the "hyper extension step" is no longer applicable. However, it is *not* obvious that this closed tableau will be derived after *finitely* many steps (i.e. it is not obvious that a refutation is order isomorphic to ω — not even for denumerable clause sets, because an inference rule might be non-continuous). Essentially, it requires to apply Königs lemma and to prove the continuity of the tableaux transforming operators.

Fortunately, this "refutational completeness" follows easily within our setup as the proof below shows. This proof makes essentially use of the fact that we view tableau construction as a *process* (derivation). An alternative approach is to define tableaux as *static* objects, which obey a closure property of branches similar to our fairness condition. This approach then requires to allow branches to be of *infinite* length, whereas we

[3] Here, "$\circ$" denotes the append function for sequences.

consider limits of branches of *finite* length. The alternative approach is attractive because it needs less formalism than our approach, and the proof of the theorem corresponding to our Theorem 15 is very simple [Hähnle, 1995]. On the other hand, we think that our formalism now pays off in order to obtain the proof of refutational completeness (without appealing to compactness, of course):

Corollary 16 (Refutational Completeness). *Let $\mathcal{S}$ be a possibly infinite, possibly non-ground, unsatisfiable clause set not containing the empty clause. Then any fair finished derivation $\mathcal{D}$ from $\mathcal{S}$ is finite and is a refutation, i.e. $\mathcal{D}$ is of the form $\mathcal{D} = T_0 \vdash_{b_0,C_0,\sigma_0,\pi_0} T_1 \cdots T_n \vdash_{b_n,C_n,\sigma_n,\pi_n} T_{n+1}$ for some $n \geq 0$, and T_{n+1} is a closed tableau.*

Proof. By Theorem 15 applied in the contrapositive direction we know that any fair derivation must contain some non-open, i.e. closed, tableau T. It is a trivial inductive consequence of our definition of "derivation" that every tableau T_i ($i \geq 0$) in a derivation contains only finitely many nodes, say $n(T_i)$, and that $n(T_{i+1}) > n(T_i)$ (the initialization step produces a tableau with finitely many nodes, and every hyper extension step applied to T_i adds only finitely many nodes to T_i, yielding T_{i+1}). Hence, for $\mathcal{D}$ to be infinite, we would have to have $(n(T_0) = 1) < n(T_1) < \cdots < n(T_n) < \cdots < n(T)$ to be an infinite chain which is impossible by well-orderedness of natural numbers. Hence $\mathcal{D}$ contains only finitely many elements.

5 Implementation

We have developed and implemented a proof procedure according to the results of the previous sections; its characteristics are, that it (a) works for the full first-order logic[4], and (b) does not backtrack over the tableaux generated along the derivation, and (c) uses universally quantified variables, and (d) employs subsumption (instead of "regularity check"). To our knowledge, no other tableaux proof procedure with these properties exists. The perhaps most advanced (non-hyper) tableaux proof procedure is the one in [Beckert and Hähnle, 1992] which uses both rigid and universal variables, but does not have property (b): if there is no refutation within a given resource bound of a maximal number of formula copies allowed for the tableaux generation, the tableau generated os far is given up, and a new one is constructed with increased bounds.

Having a proof procedure without backtracking is in particular important for the case of tableaux, because tableaux calculi usually are proof confluent (with the exception of model elimination) and so there is in principle no need for backtracking. In contrast to that, all full first-order tableaux proof procedures known to us [Fitting, 1990, Hähnle *et al.*, 1994, Beckert and Posegga, 1994, Oppacher and Suen, 1988] either employ some form of backtracking or use the γ rule to ground-instantiate the variables; we suspect the reason for this to be that no one is aware of a fairness condition which can be reasonably implemented without backtracking.

Our proof procedure is described in more detail in the long version of this paper. Here, we will only sketch the main idea. As mentioned in the introduction, the difficult

[4] By a full first order tableaux calculi we mean a calculus which uses variables at the inference level in order to abstract from terms; excluded are calculi which enumerate ground clauses, e.g. SATCHMO [Manthey and Bry, 1988] and MGTP [Fujita and Hasegawa, 1991].

issue for tableaux calculi is how to achieve fairness. We use a *weight bound* on the terms which may appear in the tableaux under construction. More precisely, the *weight* of a term (or literal) is the number of function symbols occurring in it with arity greater or equal to 1. The *weight* of a tableaux is the weight of a maximal literal occurring in it.

Now, we start with the tableau T obtained by an initialization step and initially set the weight bound w to some low value, say 1. Then, all those hyper extension steps are carried out which (1) do not violate the current weight bound, and (2) do not result in new leafs which are subsumed by the branch to be extended. Due to subsumption *and* the weight bound there is no risk of an infinite loop here. The underlying observation is that there is no infinite sequence $L_0, L_1, \ldots, L_n, \ldots$ of literals, each being lighter than w and such that L_i is *not* subsumed by some L_j, for $j < i$.

If the current weight bound w is exhausted, w is increased by some constant value (we use 1) and the next round starts with the hyper tableaux just obtained with depth bound w.

In sum, we never backtrack over the generated tableaux, and fairness is achieved by stepwisely increasing the weight bound and exhausting all hyper extension steps modulo subsumption within the given weight bound.

Improvements When interested primarily in refutational completeness (as we are) several improvements are conceivable. Currently, we implemented *factorization* and *level cut*.

By *factorization* we mean to mark an open branch $b = L_1, \ldots, L_k, \ldots, L_n$ $(k < n)$ as closed in presence of an open branch $b_L = L_1, \ldots, L_k, L$, provided that $L_n\delta = L$ for some substitution δ (and, of course, that L and L_n are labels of different nodes in case $k = n-1$). Note that δ is *not* applied to the tableaux.

The motivation for factoring is to avoid unnecessary re-derivation of subproofs; factorization and its relatives have been studied in the context of model elimination [Letz *et al.*, 1994]. What we call factorization was proposed in a similar way for SATCHMO (called "complement splitting" in [Manthey and Bry, 1988]).

For the *level cut* improvement we keep track whether an inner node is "used" to close the subtree below it. A node N is "used" in this sense if its label resolves away at least one negative literal in the SLD-refutation of at least one extending clause in the subtree below N. We can take advantage of this information *after* a closed subtree below node N is derived. Namely: if N is *not* used, then the extension step yielding N was unnecessary to obtain the closed subtree below N. As a consequence, we can think of that extension step as if it were not carried out at all and "cut off" that level, i.e. we delete all open brother nodes of N. Of course, the cancelling effect is better the more open sibling nodes are cut in this way, and the more root-most this occurs. The level cut facility was also considered as "proof condensation" in the HARP tableau prover [Oppacher and Suen, 1988].

5.1 Practical Experiments

The proof procedure of the previous section is implemented in a prototypical way as an interpreter in ECLiPSe Prolog. We ran several examples from various problem domains, and related our implementation to SATCHMO [Manthey and Bry, 1988] and OTTER. The underlying hardware is a SUN 4 workstation for all provers.

The respective entries in Figure 2 are to be read as follows: A problem identifier such as `GRP001-1` refers to its index in the TPTP problem library [Sutcliffe *et al.*, 1994]. Columns 2 – 5 contain the entries for our Hyper tableaux prover (simply called "Hyper" from now on). "L. Cut" means the level cut facility, which, as factorization, can be switched off ("-") or on ("+"). For hyper tableaux, table entries such as e.g. "1.25" and "$25+0$" in `GRP001-1` mean that the refutation took 1.25 seconds, with 25 hyper extension steps and 0 factorization steps. Blank entries mean that nothing changed with respect to the more leftmost entries.

"Range Restriction", which is mandatory for SATCHMO, means that the input clause set is transformed into range restricted form[5], whereas "Universal Variables" means that range restriction is not used.

For SATCHMO (columns 6 and 7), the "basic" version uses an incomplete depth-first search; the "level saturation" variant uses a fair strategy (this is described in [Manthey and Bry, 1988]). The numbers given are the runtimes in seconds.

OTTER (column 8) was run in "auto" mode, where it analyzes the input clause set and determines inference rules and strategies by itself. In most examples, positive hyper resolution was the inference rule chosen automatically, possibly augmented by a completion-based equality handling. However, since we are mainly interested in the relationship Hyper tableaux vs. hyper resolution, we had been unfair to OTTER in a few cases and forced positive hyper resolution without a dedicated equality reasoning. The entries give the runtimes in seconds (such as "0.1" for `GRP010-1`), and, in the subsequent row (such as "5" for `GRP010-1`), the number of clauses kept in the refutation. The values in parenthesis are the results where backward subsumption is switched *off*. We are aware that this is again unfair to OTTER, but it supports a direct comparison between the Hyper tableaux and hyper resolution calculi.

Let us comment on the results in Figure 2. We distinguish four groups, which are horizontally separated by double lines; we proceed from top to bottom.

Propositional and Horn Problems. Since the calculi underlying the three provers are the same, we can use such examples to evaluate the quality of implementation. As probably is to be expected, OTTER (written in C) is about eight times as fast as Hyper (being an interpreter written in Prolog). The good results for SATCHMO can be explained by a better usage of the built-in term retrieval primitives (assert/retract).

Propositional and non-Horn Problems. For the *unsatisfiable* problems (`MSC007-1`) the timing results for Hyper are close to that of OTTER without backward subsumption (this is unlike to the previous group). We emphasize that the runtimes for Hyper are *not* normalized. SATCHMO performs well for the reason stated.

For the *satisfiable* `SYN091` examples we can observe a real advantage of the tableaux approach. Hyper (as well as SATCHMO) immediately stops as soon as one branch is

[5] A clause is *range restricted* iff every variable occurring in the head also occurs in the body; every clause set can trivially be transformed into range restricted form, see [Manthey and Bry, 1988].

Problem	Hyper Tableaux: Domain Restriction +L. Cut +Factor	Hyper Tableaux: Domain Restriction -L. Cut -Factor	Hyper Tableaux: Universal Variables +L. Cut +Factor	Hyper Tableaux: Universal Variables -L. Cut -Factor	SATCHMO Dom. Restr. Basic	SATCHMO Dom. Restr. Lev.Sat.	OTTER Auto
`SYN089-1.005`	0.8				**0.1**	**0.1**	0.4(0.4)
	26 + 0						111(111)
`SYN089-1.010`	21				1.5	**0.9**	2.8(2.8)
	101 + 0						471(471)
`SYN089-1.015`	123				6.4	**3.0**	15.2(15.2)
	226 + 0						1081(1081)
`MSC007-1.004`	0.4				**0.1**	0.2	0.2(0.2)
Pigeonh. 4in3	49 + 0						49(49)
`MSC007-1.005`	3.5				2.0	3.9	**1.3**(3.6)
Pigeonh. 5in4	261 + 0						187(187)
`MSC007-1.006`	218				36	81	**15**(314)
Pigeonh. 6in5	1631 + 0						952(952)
`SYN091-1.005`	0.4				**0.1**	**0.1**	17(28)
(satisfiable)	9 + 0						1538(1537)
`SYN091-1.010`	4.1				0.5	**0.4**	$> 0.5h$
(satisfiable)	19 + 0						
`GRP001-1`	1.25		0.35		$> 0.5h$	$> 0.5h$	**0.1**(**0.1**)
(Group Th.)	25 + 0		11 + 0				5(5)
`GRP010-1`	156		58		$> 0.5h$	$> 0.5h$	**0.3**(0.4)
(Group Th.)	179 + 0		92 + 0				42(42)
`GRP013-1`	$> 0.5h$		$> 0.5h$		$> 0.5h$	$> 0.5h$	**10**(7.5)
(Group Th.)			92 + 0				2122(2122)
`MSC006-1`	0.9	13.2	**0.8**	11.5	1.9	2230	3.6(3.7)
(NonObv)	57 + 6	507 + 0	53 + 6	503 + 0			70(70)
`PRV002-1`	131	$> 0.5h$	13	215	$> 0.5h$	$> 0.5h$	**1.0**(**1.0**)
(Progr.Verif.)	490 + 0		89 + 0				183(183)
`PUZ001-2`	31	61	11	14	$> 0.5h$	$> 0.5h$	3.4(**3.1**)
(Pelletier55)	226 + 1	359 + 0	104 + 1	117 + 0			546(546)
`PUZ005-1`	286	$> 0.5h$	3.7	1837	$> 0.5h$	$> 0.5h$	**2.0**(2.9)
(Lion+Unic.)	628 + 35		104 + 13	4957 + 0			255(242)
`PUZ023-1`	151	405	0.6	0.8	16	$> 0.5h$	**0.3**(**0.3**)
(Knights+Kn.)	124 + 0	159 + 0	31 + 3	48 + 0			43(43)
`PUZ024-1`	9.9		0.2		**0.1**	$> 0.5h$	**0.1**(**0.1**)
(Knights+Kn.)	60 + 0		16 + 0				16(16)
`PUZ025-1`	672	2321	3.1	14	7.8	$> 0.5h$	**1**(**1**)
(Knights+Kn.)	173 + 4	605 + 0	67 + 3	329 + 0			131(131)
`PUZ026-1`	10.4	10.9	0.8	0.9	**0.1**	$> 0.5h$	19(27)
(Knights+Kn.)	68 + 14	82 + 0	33 + 9	43 + 0			203(212)
`PUZ030-1`	2.8	2.3	1.8	**1.4**	7.5	720	50($> 0.5h$)
(Salt+Must.)	105 + 10	119 + 0	100 + 10	114 + 0			518
Steamroller	2.1		0.9		1.4	52	**0.7**(1)
(SATCHMO)	46 + 0		28 + 0				68(68)
Steamroller*	12.3	51.0	5.5	22	1.4	$> 0.5h$	**1**(**1**)
	113 + 1	590 + 0	71 + 1	320 + 0			146(146)

Fig. 2. Runtime results for our hyper tableau prover, SATCHMO and OTTER.

finished and reports the model. OTTER will not recognize the satisfiability that early and continues building hyper resolvents.

First-Order Horn Problems. The results in columns 2 and 3 ("domain restriction") vs. columns 4 and 5 ("universal variables") demonstrate the superiority of the universal variables approach within Hyper tableaux. For the stated examples, it prevents the prover from enumerating instances of the reflexivity axiom $X = X$; instead it extends with $X = X$ in the first step, and then subsumes all possibly upcoming instances of it.

The Hyper prover in the domain restriction setting is comparable to SATCHMO. The superiority of Hyper to SATCHMO in this case might be our weight-bounded enumeration of literals. The rationale for this strategy is the assumption that if a refutation is within the possibilities of our prover at all, then it will be discovered at a shallow term level. This is because as the weight bound gets too heavy, far too many terms will be generated in purifying substitutions and the prover gets lost.

The success of OTTER compared to Hyper (cf. `GRP013-1`) can be explained by using a more clever weighting function.

First-Order non-Horn Problems. Hyper performs well on all examples. Notice that in many cases SATCHMO fails to find a refutation, in particular if the complete level saturation strategy is employed. When relating the timing results of OTTER to that of Hyper, one should keep in mind that there is a factor of eight to be observed due to the quality of implementation (cf. the first group). If normalized, the results for Hyper would be better than that of OTTER in almost all cases. Furthermore, as was also argued for in the previous group, OTTER seems to use a more clever weighting function. This lets us speculate that Hyper can be improved significantly by learning the weighting function from OTTER.

In this problem group, the calculi underlying the three provers deviate significantly. The effects mentioned in the previous group apply here as well, but in an even more drastic way. For instance, the enumeration of ground instances of $X = X$ in `PUZ001-2`[6] will happen in every branch (Hyper with domain restriction and SATCHMO). By enumerating the ground instances a higher local search space for the SLD-resolution in hyper extension steps results. This might be one of the keys to the success of the universal variables. This claim is supported by almost all of the puzzle examples and `PRV002-1`. A counterexample is `MSC006-1`, but here the set of ground instances is small (four constants, no function symbols).

Next we discuss the merits of *factorization* and *level cut*. For this, column 3 (resp. 5) has to be compared to column 2 (resp. 4). There are several examples where these techniques turn out to be useful, with the most striking case being `PUZ005-1`. Here, a seven place disjunction $monday(X) \vee \cdots \vee sunday(X)$ is present, which can be used for extension steps at almost every time. Cutting off useless applications of this clause is most effective in this case. Another example where this applies is "steamroller". In the "steamroller*" version (taken from [Loveland *et al.*, 1995]), a redundant clause

$$animal(X),\ animal(Y) \rightarrow quicker(X,Y)\ ; smaller(Y,X)$$

[6] [Beckert and Hähnle, 1992] report on a proof of this problem in about 4 seconds, but their prover has special inference rules for equality.

is added, which results in many useless case analyses. Our prover solves this by the level cut facility; the SATCHMORE prover [Loveland *et al.*, 1995] solves this problem by a relevance analysis ("*quicker*" is pure in the input clause set). It should be noted that the other drastic example from [Loveland *et al.*, 1995] (Example 19) which demonstrates the usefulness of the relevance analysis can also be solved in about 1 second with our prover when the level cut is employed.

We switched the two flags individually in all four combinations, but did not report on the results in Figure 2. Instead we summarize our observations that the *level cut* is far superior to the factorization rule. All problems considered by us can be computed in almost the same (quite often even in shorter time) if level cut is used alone.

To summarize our experiments, we think that the design of our calculus and proof procedure results in a significant improvement of bottom-up, model-based theorem proving.

6 Conclusion

We presented the *hyper tableau* calculus, which combines ideas from resolution (subsumption, universal variables) with analytic tableaux. We obtained a completeness result which allows for a reasonable procedure without backtracking over the generated tableaux. We demonstrated its practical usefulness using examples from various problem domains. We are aware that the calculus/proof procedure can still be considerably improved by lifting the ground terms generated in purifying substitutions to rigid variables. Another interesting improvement is proposed by [Billon, 1996] within his disconnection method. This is a proof confluent calculus, which extends the similarity to resolution with respect to universally quantified even more. Translated into our framework the idea is to avoid rigid variables by extending a branch not only with an input clause but additionally by appropriate instances of the "other parent clauses". If the input clauses contain $p(x), q(x) \leftarrow$ and $r \leftarrow p(f(y))$ and a tableau is constructed which contains a branch with $p(x)$ stemming from the disjunctive fact, an extension with $r \leftarrow p(f(y))$ is possible. But additionally the instance $p((f(y)), q(f(y)) \leftarrow$ of $p(x), q(x) \leftarrow$ has to be fanned below $p(x)$ before the extension with $r \leftarrow p(f(y))$ is carried out. We are currently adapting this idea to hyper tableaux and its implementation.

An intersting relation to SLO-resolution was pointed out by an anonymous referee. As it is introduced in [Rajasekar, 1989] SLO-resolution is a goal oriented calculus for positive disjunctive programs which is presented as an extension of SLD-resolution. If all literal signs from the program clauses and from the goal clause are complemented (which preserves satisfiability) our hyper tableaux calculus corresponds to SLO-resolution. It is exactly the case for ground derivations, whereas in non-ground cases our calculus is an extension of SLO-resolution. A detailed investigation of this topic can be found in [Baumgartner and Furbach, 1996].

Finally it is worth mentioning, that this kind of model generation by tableau calculi is very well suited for the construction of minimal models, and hence for non-monotonic reasoning. In [Niemelä, 1996b, Niemelä, 1996a] a variant of hyper tableaux is used to compute minimal model entailment of negated atoms and in [Bry and Yaha, 1996] a formalization of SATCHMO is used to derive minimal models.

References

[Baumgartner and Furbach, 1996] Peter Baumgartner and Ulrich Furbach. Hyper Tableaux and Disjunctive Logic Programming. Fachberichte Informatik 13–96, Universität Koblenz-Landau, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 1996.

[Beckert and Hähnle, 1992] B. Beckert and R. Hähnle. An Improved Method for Adding Equality to Free Variable semantic Tableaux. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *LNCS*, pages 507–521. Springer, 1992.

[Beckert and Posegga, 1994] Bernhard Beckert and Joachim Posegga. leanT^AP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 1994. To appear.

[Billon, 1996] Jean-Paul Billon. The Disconnection Method. In P. Moscato, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, volume 1071 of *Lecture Notes in Artificial Intelligence*. Springer, 1996.

[Bry and Yaha, 1996] F. Bry and A. Yaha. Minimal model generation with unit hyper-resolution tableau. In *5th Workshop of Analytic Tableaux and Related methods*, LNCS. Springer, 1996.

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

[Fitting, 1990] M. Fitting. *First Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer, 1990.

[Fujita and Hasegawa, 1991] H. Fujita and R. Hasegawa. A Model Generation Theorem Prover in KL1 using a Ramified-Stack Algorithm. In *Proc. of the Eigth International Conference on Logic Programming*, pages 535–548, Paris, France, 1991.

[Hähnle *et al.*, 1994] R. Hähnle, B. Beckert, and S. Gerberding. The Many-Valued Theorem Prover 3TAP. Interner Bericht 30/94, Universität Karlsruhe, 1994.

[Hähnle, 1995] R. Hähnle. Positive tableaux. Research note, 1995.

[Letz *et al.*, 1994] R. Letz, K. Mayr, and C. Goller. Controlled Integrations of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13, 1994.

[Loveland *et al.*, 1995] D. Loveland, D. Reed, and D. Wilson. SATCHMORE: SATCHMO with RElevance. *Journal of Autmated Reasoning*, 14:325–351, 1995.

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

[Niemelä, 1996a] I. Niemelä. Implementing circumscription using a tableau method. In *Proceedings of the European Conference on Artificial Intelligence*, Budapest, Hungary, August 1996. John Wiley. To appear.

[Niemelä, 1996b] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294, Terrasini, Italy, May 1996. Springer-Verlag.

[Oppacher and Suen, 1988] F. Oppacher and E. Suen. HARP: A Tableau-Based Theorem Prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

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

[Robinson, 1965] J. A. Robinson. Automated deduction with hyper-resolution. *Internat. J. Comput. Math.*, 1:227–234, 1965.

[Siekmann, 1989] Jörg H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7(1):207–274, January 1989.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In A. Bundy, editor, *Proc. CADE-12*, volume 814 of *LNAI*. Springer, 1994.

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