Merge Path Improvements for Minimal Model Hyper Tableaux

Peter Baumgartner, J.D. Horton, Bruce Spencer
University of New Brunswick
Fredericton, New Brunswick E3B 5A3, Canada
{baumgart,jdh,bspencer}@unb.ca

December 21, 1998

Abstract

We combine techniques originally developed for refutational first-order theorem proving within the clause tree framework with techniques for minimal model computation developed within the hyper tableau framework. This combination generalizes well-known tableaux techniques like complement splitting and folding-up/down. We argue that this combination allows for efficiency improvements over previous, related methods. It is motivated by application to diagnosis tasks; in particular the problem of avoiding redundancies in the diagnoses of electrical circuits with reconvergent fanouts is addressed by the new technique. In the paper we develop as our main contribution in a more general way a sound and complete calculus for propositional circumscriptive reasoning in the presence of minized and varying predicates.

1 Introduction

Logical reasoning systems solve two problems: (1) given a formula, find a model (or situation in which the formula is true) or (2) determine that no model exists. When solving the first problem, finding models, it is important to find a minimal model, or situation that makes no unnecessary assumptions. These models can be applied to problems in programming language semantics, knowledge representation and diagnosis. The second problem is equivalent to automated theorem proving, where one refutes the complement of theorem to be proved. In this paper we concentrate on the first problem.

Recently clause trees [Horton and Spencer, 1997a], a data structure and calculus for automated theorem proving, introduced a general merge operation to eliminate open goals in the tree, based on so-called *merge paths* (cf. Section 1.1). In essence the clause tree allows one to build just one tree, but it implicitly represents a (usually large) number of different binary resolution proofs. Merge paths are related to the folding-up and folding-down inference rules developed within model elimination [Letz *et al.*, 1994]. In effect, merge paths realize their combination. This combination is not trivial, as care must be taken not to lose soundness.

Tableau calculi of the "hyper" type for minimal model reasoning were proposed in [Bry and Yahya, 1996; Niemelä, 1996a]. The most general variant is [Niemelä, 1996b] which computes circumscription in the presence of minimized, fixed and varying predicates targetting at first-order logic.

The *merge path* technique suggested in the present paper is intended as an additional inference rule to avoid certain redundancies in this type of calculi. We use our own framework of *hyper tableau* for this, which began with [Baumgartner *et al.*, 1996]. A more recent first-order version is described in [Baumgartner, 1998]. Concerning minimal model computation, ground versions of the hyper tableau calculus were used in [Baumgartner *et al.*, 1997a; Baumgartner *et al.*, 1997b] for diagnosis applications, and in [Aravindan and Baumgartner, 1997] for view updates in deductive databases.
It is common to all these model computation calculi that they use *atomic cuts* (cf. Lemma 5.1 in Section 5) explicitly or – slightly weaker[1] – implicitly built-in into inference rules to speed up derivations. These usages closely correspond to a special case of merge paths called *right hooks*, which is also known as folding-down in [Letz *et al.*, 1994] or complement splitting in the Satchmo family of provers, which traces back to [Manthey and Bry, 1988]. Essentially, complement splitting replaces a disjunction $A \vee B$ by $(A \wedge \neg B) \vee B$.

In this paper we advocate to use more general merge paths of [Horton and Spencer, 1997a] for model computation calculi. This allows branches to close earlier than it would be possible without merge paths or when using merge paths to simulate known instances such as folding-down. One generalization is the case of *left hooks*. They are symmetrical to right hooks. Expressed from the viewpoint of complement splitting, the advantage is that the splitting of literals can be *deferred*. In the example, it might become only clear later in the derivation that splitting towards $A \vee (B \wedge \neg A)$ is more fruitful. This is accomplished by the device of left/right hooks.

Besides hooks there is another kind called *deep merge paths* which correspond to folding-up. They have to be applied with care, as they can introduce some computational overhead. This holds in particular when computing models is demanded (as opposed to refutational theorem proving). The paper [Horton and Spencer, 1997a] is devoted to refutational theorem proving, and the results from there do not directly carry over to our task of computing minimal models (nethertheless, many general results about cyclic dependencies of paths in [Horton and Spencer, 1997a] can readily be taken for our case). The same can be said about the folding-up/down inference rules of [Letz *et al.*, 1994] which were (a) proposed for refutational theorem proving and (b) are not used together (in order to avoid the check for cyclic dependencies).

Therefore, we give conditions such that the central properties of minimal model soundness and minimal model completeness hold. More precisely, as our main result we develop such a calculus for the more general case of circumscriptive reasoning (Section 6).

The minimal model completeness proof is given by a simulation of merge paths by

[1] This is slightly weaker, because the presence of negative literals as introduced by the cut rule, as in [Niemelä, 1996b], can be used to trigger the derivation of new negative unit clauses, which cause no branching in the search space ("affirmative negative rule" in the Davis-Putnam procedure.)

atomic cuts (cf. Lemma 5.1 in Section 5). Viewed from this point, our approach can thus be seen as one more and generalized approach for a *controlled* integration of the cut rule for the purpose of minimal model computation.

The rest of this paper is structured as follows: first we briefly give the idea of *merge paths* as defined in [Horton and Spencer, 1997a]. This presentation should be sufficient to explain the subsequent motivation of the new calculus from the viewpoint of a certain problem encountered in diagnosis tasks. In Section 2 we bring in merge paths into trees and define an ordering on merge paths. It is employed in Section 3 in a first version of the new calculus. In Section 4 the problem of possible nontermination is explained, and as a solution a sufficient termination condition is proposed. With this prerequisites, we can give soundness and a first minimal model completeness result in Section 5. After that, the calculus is be generalized in Section 6 to a circumscription variant. Furthermore, additional conditions are defined so that minimal model soundness holds as well. Finally, we conclude in Section 7.

Question: How costly is it to check for legality? I guess the cost of checking that $\mathcal{P} \cup \{p\}$ is legal provided that $\mathcal{P}$ is legal is linear in the number of nodes.

1.1 Clause Trees

Merge paths are introduced and extensively studied in [Horton and Spencer, 1997a] in the context of clause trees. Clause trees are a data structure that represent equivalence classes of resolution derivations.

Merge paths serve as unified inference rule and generalize the folding up/folding down technique of [Letz *et al.*, 1994]. Due to space reasons we only try to explain clause trees and merge paths using a simple example.

Figure 3 depicts a clause set together with a clause tree proof. Clause trees consist of *clause* nodes and *atom* nodes. Clause nodes are indicated by a $\circ$. Every clause node N corresponds to some input clause $\lambda(N) = L_1 \vee \ldots \vee L_n$ as can be seen from the n emerging edges; these edges are labeled by the signs of the L_i's, and the atom parts of the L_i's can be found in the adjacent atom nodes. Clause trees are built in such a way that from every atom node exactly two edges with opposite sign emerge. This corresponds to a binary resolution inference. A proof is a clause tree where every leaf is an atom node.

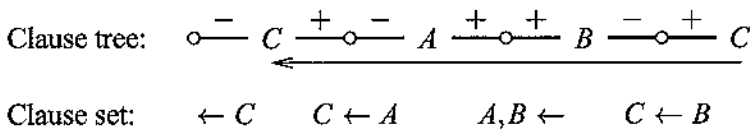

Figure 1: An example for a clause tree with merge path. It corresponds to the tree in Figure 3, except that a different input clause set is used.

Now, in addition, *merge paths* can be drawn between equally labeled atom nodes. In Figure 3 there is a merge path from the right C-node (called the *tail* of the merge

path), to the left C-node (called the *head* of the merge path). The idea is "in order to find a proof at the tail of a merge path, look it up (copy it) from the head of the merge path". Thus, tail nodes are considered as proven as well and need no further extension. Head nodes can be part of another merge path. In this case the "lookup" of proofs is done recursively. In order to terminate this, cyclic dependencies must be excluded. The absence of cycles in a set of merge paths is referred to by the term "legal". Many of the results in [Horton and Spencer, 1997a] concerning legality and relation notions are derived as general properties of paths in trees. They thus can be readily applied to our case of hyper tableaux as well.

1.2 Motivation: A Diagnosis Application

We consider consistency-based diagnosis according to Reiter [Reiter, 1987]. In this scenario, a model of a device under consideration is constructed and is used to predict its normal behavior. By comparing this prediction with the actual behavior it is possible to derive a diagnosis. More precisely, a *diagnosis* Δ is a subset of the components of the device, such that the observed behavior is consistent with the assumption that exactly the components in Δ are behaving abnormally. Usually, one is interested in subset-minimal diagnoses. Also, the cardinality of Δ is usually restricted to small values (even $|\Delta| = 1$ is interesting).

Of course, all this can be formalized properly within first-order or even propositional logic (see [Reiter, 1987]), and computing diagnoses becomes a circumscription task. In [Baumgartner *et al.*, 1997b] it was shown that a "standard" hyper tableau based theorem prover is competitive to the DRUM-2 dedicated diagnosis engine [Nejdl and Fröhlich, 1996].

Figure 2 depicts a hypothetical diagnosis scenario of an electrical circuit. The notation [0] in the left picture means that at this point the circuit is logical zero. The [0]'s at the bottom refer to input values of the actually observed behavior, but it is also conceivable that these are passed in from other parts of the circuit below them. The "Huge" box is meant to stand for a large circuit. The lightning at the output indicates that the predicted output is different from the actual output. Among all possible diagnoses we concentrate on $\Delta_1 = \{inv1\}$ and $\Delta_2 = \{inv2\}$. Notice that with declaring these as "abnormal", it is consistent to have [0] at the output of the *and*-gate, and we suppose that this is essential to render the whole system consistent then.

Now, the crucial observation is that the computation of Δ_1 and Δ_2 show considerable redundancies. The hyper tableau based diagnosis approach of [Baumgartner *et al.*, 1997b] would result in the tableau in the middle of Figure 2[2]. Diagnoses are read off from open branches by collecting the *ab*-literals found there. The triangles stand for sub-tableaux containing diagnoses of the "Huge" part. There are two open branches containing Δ_1 and Δ_2 respectively.

Notice that the "Huge" part has to be diagnosed twice *although for its diagnosis*

[2]To be precise, this is not quite true, as the diagnosis approach in [Baumgartner *et al.*, 1997b] as well as in [Nejdl and Fröhlich, 1996] is semantically guided and would attempt a goal-directed diagnosis starting from the output of the circuit. However, the symmetrical problem would occur there.

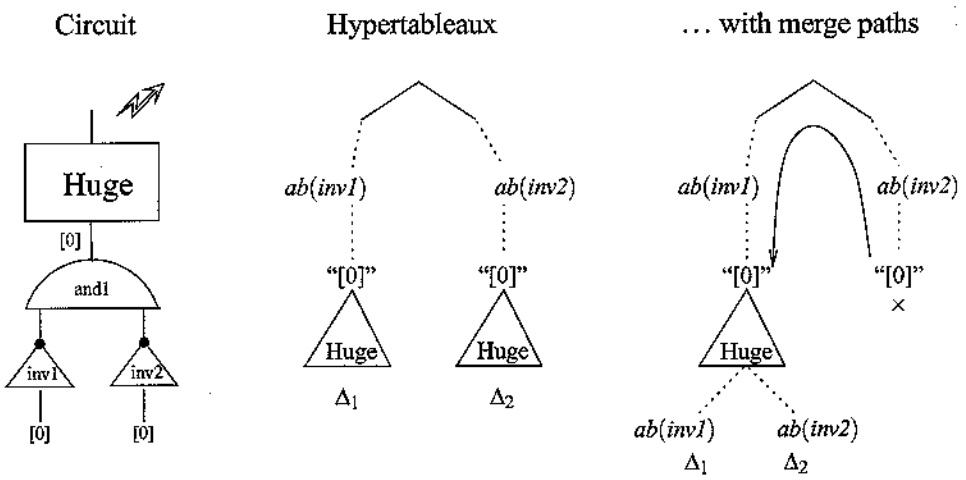

Figure 2: A diagnosis scenario where merge paths are useful. $\Delta_1 = \{inv1\}$ and $\Delta_2 = \{inv2\}$ are two diagnoses.

exactly the same situation applies, namely [0] *at its input.* This is reflected by the nodes "[0]". Clearly, for the diagnosis of "Huge" it is irrelevant what caused the "[0]"-situation. The generalized underlying problem is well-known in the diagnosis community and is referred to as "reconvergent fanouts".

So, the symmetry hidden in this problem was not exploited. In fact, the merge path technique just realizes this. It is indicated in the right part of Figure 2: after the diagnosis Δ_1 is computed in the left branch, and the computation reaches the "[0]" node in the right subtree, a merge path is drawn as indicated, and the branch with the right "[0]" node is closed. The price to be paid is that Δ_1 as computed so far is invalid now. Technically, the $ab(inv1)$ literal can be thought of as being removed from the branch (it becomes "invisible" in our terminology). Hence, the computation starts again as indicated below the triangle. Eventually, both Δ_1 and Δ_2 can be found there.

Why is it attractive to use such a "non-monotonic" strategy? The answer is that it is little effort to recompute the initial segment of the diagnosis and better to save recomputing the "huge" part. We do not suggest to use the merge paths in all possible situations. In order to be flexible and allow guidance by heuristics, merge paths are thus always *optional* in the calculi defined below.

1.3 Preliminaries

We assume that the reader is familiar with the basic concepts of first-order logic. Throughout this paper, we are concerned with propositional logic[3]. A *clause* is an expression of the form $\mathcal{A} \leftarrow \mathcal{B}$, where $\mathcal{A} = (A_1, \ldots, A_m)$ and $\mathcal{B} = (B_1, \ldots, B_n)$ are finite sequences of atoms $(m, n \geq 0)$; $\mathcal{A}$ is called the *head*, and $\mathcal{B}$ is called the *body* of the clause. Whenever

[3]To be precise, we consider variable-free (ground) first-order clausal logic. Furthermore, only finite ground clause sets are considered.

convenient, a clause is also identified with the disjunction $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ of literals.

Quite often, the ordering of atoms does not play a role, and we identify $\mathcal{A}$ and $\mathcal{B}$ with the sets $\{A_1,\ldots,A_m\}$ and $\{B_1,\ldots,B_n\}$, respectively. Thus, set-theoretic operations (such as "$\subset$", "$\cap$" etc.) can be applied meaningfully.

By $\overline{L}$ we denote the complement of a literal L. Two literals L and K are *complementary* if $\overline{L} = K$.

In the sequel, the letters K and L always denote literals, A and B always denote atoms, C and D always denote clauses, $\mathcal{S}$ always denotes a finite ground clause set, and Σ denotes its signature, i.e. $\Sigma = \bigcup\{\mathcal{A} \cup \mathcal{B} \mid \mathcal{A} \leftarrow \mathcal{B} \in \mathcal{S}\}$.

As usual, we represent a Σ-interpretation $\mathcal{I}$ by the set of *true* atoms, i.e. $\mathcal{I}(A) = \mathit{true}$ iff $A \in \mathcal{I}$. Define $\mathcal{I} \models \mathcal{A} \leftarrow \mathcal{B}$ iff $\mathcal{B} \subseteq \mathcal{I}$ implies $\mathcal{A} \cap \mathcal{I} \neq \emptyset$. Notice that this is consistent with other usual definitions when clauses are treated as disjunctions of literals. Usual model-theoretical notions of "satisfiablity", "validity" etc. of clauses and clause sets are applied without defining them explicitly here.

2 Literal Trees and Merge Paths

As a new contribution of this paper, we use the "path" device of the previous section and bring it to hyper tableaux in Section 3. Before doing so we need some notions and preliminary definitions which are introduced in this section.

We consider finite ordered trees T where the nodes, except the root node, are labeled with literals. Let λ be the labeling function. A *branch* (of T) is a sequence $b = (N_0, N_1, \ldots, N_n)$ of nodes of T such that N_0 is the root, N_i is an immediate successor node of N_{i-1} (for $1 \leq i \leq n$) and N_n is a leaf node. The fact that b is a branch of T is also written as $b \in T$.

Any subsequence $b' = (N_i,\ldots,N_j)$ with $0 \leq i \leq j \leq n$ is called a *partial branch of* b; if $i = 0$ then this subsequence is called *rooted*. Define $\mathit{last}(b') = N_j$. In the sequel the letter b always denotes a branch or a partial branch.

The expression (b_1, b_2) denotes the concatenation of partial branches b_1 and b_2; similarly, the expression (b,N) denotes $(N_i,\ldots,N_j,N)$, where b is the partial branch $(N_i,\ldots,N_j)$.

For convenience we write "the node L", where L is a literal, instead of the more lengthy "the node N labeled with L", where N is some node given by the contex. In the same spirit, we write $(L_1,\ldots,L_n)$ and mean the partial branch $(N_1,\ldots,N_n)$, or even $(N_0,N_1,\ldots,N_n)$ in case N_0 is the root and N_i is an immediate successor node of the root, where N_i is labelled with L_i (for $1 \leq i \leq n$). Further, (b,L) means (b,N), where N is some node labeled with L and b is a partial branch.

In order to remember the fact that a branch contains a contradiction, we label branches as either *open* or *closed*. A tableau is *closed* if each of its branches is closed, otherwise it is *open*. This notion is also applied to subtrees.

Definition 2.1 (Ancestor Path, Merge Path)
Let T be a tree and $\mathcal{P}$ be a set of merge paths. Suppose that T contains a rooted partial branch b of the form $b = (N_0, N_1, \ldots, N_i, \ldots, N_n)$ with N_0 being the root. Any sequence

$$ancp(b, N_i) := (N_n, N_{n-1}, \ldots, N_i) \text{ , where } n \geq i > 0$$

is called an *ancestor path (of b)*. The node N_n is called the *tail* and the node N_i is called the *head* of this ancestor path. Now, if it additionally holds that $\lambda(N_i) = A$ and $\lambda(N_n) = \neg A$ (for some atom A) then $ancp(b, N_i)$ is called an *ancestor merge path (of b)*.

Suppose that T contains two different rooted partial branches of the form $b^T = (N_0, N_1, \ldots, N_i, N_{i+1}, \ldots N_n)$ and $b^H = (N_0, N_1, \ldots, N_i, M_{i+1}, \ldots M_m)$ with N_0 being the root and $m, n > i \geq 0$ and $M_{i+1} \neq N_{i+1}$ and such that $\lambda(N_n) = \lambda(M_m) = A$ for some atom A. Define

$$\begin{aligned} p^T &= N_n, \ldots, N_{i+1} \\ p^H &= M_{i+1}, \ldots, M_m \\ p &= (p^T, p^H) \ . \end{aligned}$$

Here, p is understood as a concatenation of p^T and p^H. We assume that p can always be decomposed into its constituents p^T and p^H; p is called a *non-ancestor merge path of T from b^T to b^H with tail N_n and head M_m*. It is also denoted by $mergep(b^T, b^H)$[4]. The node N_i is called the *turn point* of p.

By a *merge path* we mean a non-ancestor merge path or an ancestor merge path. For any merge path or ancestor path $(N_1, \ldots, N_n)$, every node $N_2, \ldots, N_{n-1}$ is called an *inner node*.

The letters p and q are used in the sequel to denote ancestor paths or merge paths. □

Some special cases of non-ancestor merge paths: the case $m = n = i + 1$ is called *factoring*, the case $m = i + 1$ is called a *hook*, and the case $m > i + 1$ is called a *deep merge path*.

Below we take advantage of merge paths in inference rules to close branches. Then, *hooks*, when applied in a special way can be used to simulate folding down and complement splitting, and deep merge paths, also when applied in a special way, can be used to simulate *folding up*. Usually, only either folding down/complement splitting or folding up is applied. The reason is that an unrestricted application of these inference rules yields an unsound calculus. In order to arrive at a sound calculus, one has to avoid certain circular dependencies. The following definition is intended to make this more precise.

Definition 2.2 (Ordering on paths [Horton and Spencer, 1997a])
Let $p = (N_1, \ldots, N_n)$ and $q = (M_1, \ldots, M_m)$ be paths. Define *q precedes p*, as

$$q \prec p \text{ iff } M_m \in \{N_2, \ldots, N_{n-1}\} \ .$$

Following [Horton and Spencer, 1997a] we say that a finite set of paths $\mathcal{P}$ is *legal* iff the $\prec$ relation on $\mathcal{P}$ can be extended to a partial order $\ll$ on $\mathcal{P}$. The term "illegal" means "not legal". □

[4]Due to the requirement $M_{i+1} \neq N_{i+1}$ it is also unique, for given b^T and b^H

Notice that the $\prec$ relation is irreflexive but in general not transitive.

Equivalently to the previous definition, one could also define a set of paths to be illegal if it contains a cycle, i.e. if there are paths $p_1,\ldots,p_n \in \mathcal{P}$ such that $p_1 \prec p_2 \prec \cdots \prec p_n \prec p_1$, for some $n > 1$.

Example 2.3 (Ordering) Figure 3 contains examples of some trees equipped with merge paths. The underlying clause sets can be left implicit. Merge paths are indicated using arrow notation. For instance, in the right tree, the arrow from the leaf node $\neg A$ to A indicates an (the) ancestor merge path $p_1 = (A,C,\neg A)$ of the branch $(A,C,\neg A)$ with tail $\neg A$ and head A. In the same tree, the arrow from the rightmost node C to the other node C indicates a non-ancestor merge path $p_2 = mergep((B,C),(A,C)) = (C,B,A,C)$ with tail C (the right node) and head C (the other node C) and the root as turn point. In terms of Definition 2.1 we have $p^T = (C,B)$ and $p^H = (A,C)$. The path p_2 is an example of a *deep merge path*. The merge path set $\{p_1,p_2\}$ is not legal because both $p_1 \prec p_2$ and $p_2 \prec p_1$ and hence $\prec$ cannot be extended to a partial order.

The left tree in Figure 3 contains two non-ancestor merge paths and both are "hooks". The left one with head B is an example of a "right hook" and the other one is a "left hook" in the terminology of [Horton and Spencer, 1997a].

The left and right cases are the simplest cases for illegality, as in both cases only two merge paths are involved. These are illegal, because the heads of the merge paths are mutually contained as inner nodes. Of course, the situation can be more complicated by producing such a cycle with more than two merge paths.

The reader might wish to return to this example after having read the definition of the *hyper tableau calculus with merge paths* (Def. 3.1). □

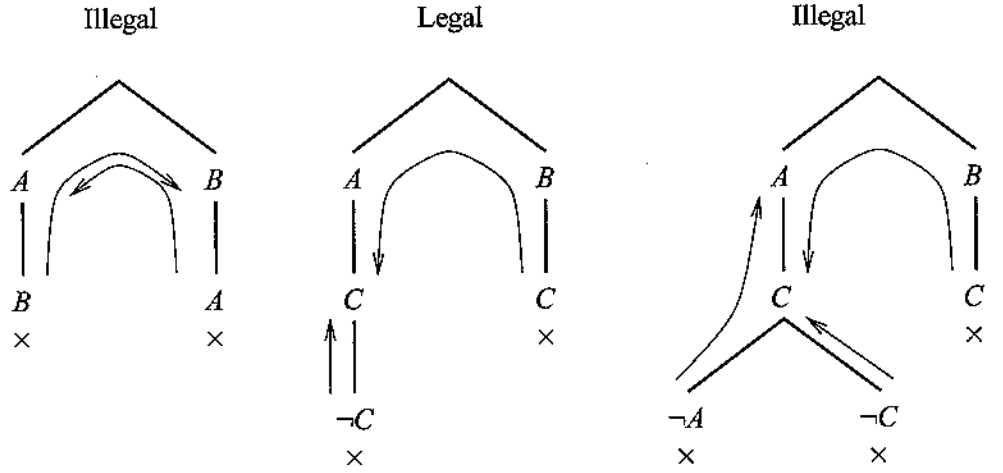

Figure 3: Examples for legal and illegal merge paths.

The new calculus to be presented below does not only construct a tableaux T as the derivation proceeds, but also a set of merge paths $\mathcal{P}$. When interested in refutational theorem proving only, one would only have to be careful not to construct illegal path

sets. However, we are interested in a more general minimal model completeness result. In order to achieve this, we have to define how interpretations are extracted from open branches.

Definition 2.4 (Visibility, Branch Semantics)
Let $b = (N_0, N_1 \ldots, N_n)$ be a rooted partial branch in a tree T (not necessarily a hyper tableau) with $n \geq 0$, and let $\mathcal{P}$ be a set of legal merge paths. The node N_i (where $0 < i \leq n$) which is not the tail of a merge path in $\mathcal{P}$ is said to be *visible from* N_n *in* $\mathcal{P}$ iff $\mathcal{P} \cup \{ancp(b, N_i)\}$ is legal. Define

$$[\![(N_0, N_1, \ldots, N_n)]\!]_{\mathcal{P}} = \{\lambda(N_i) \mid N_i \text{ is visible from } N_n \text{ wrt. } \mathcal{P}, \text{ for } 0 < i \leq n\} \ .$$

□

For instance, in the middle tableau in Figure 3 we have $[\![(A, C, \neg C)]\!]_{\mathcal{P}} = \{\neg C, C\}$ and $[\![(B, C)]\!]_{\mathcal{P}} = \{B, C\}$, where $\mathcal{P}$ consists of the two merge paths drwn there. Notice that the case $n = 0$ is not excluded, and it holds that $[\![(N_0)]\!] = \emptyset$. Notice further that if $[\![b]\!]_{\mathcal{P}}$ does not contain negative literals then $[\![b]\!]_{\mathcal{P}}$ is an interpretation. This holds in particular for the "open" branches computed by our calculi below.

Lemma 2.5 (Basic properties)
For any tree T with open branch b and legal path set $\mathcal{P}$ the following holds:

(i) For every ancestor path p such that $\mathcal{P} \cup \{p\}$ is legal, $[\![b]\!]_{\mathcal{P} \cup \{p\}} = [\![b]\!]_{\mathcal{P}}$.

(ii) For every set of ancestor paths $\mathcal{P}'$ such that $\mathcal{P} \cup \mathcal{P}'$ is legal it holds $[\![b]\!]_{\mathcal{P} \cup \mathcal{P}'} = [\![b]\!]_{\mathcal{P}}$.

Proof. **(i).** By definition $A \in [\![b]\!]_{\mathcal{P}}$ iff some node N_A of b labeled with A is visible from the leaf $last(b)$ of b in $\mathcal{P}$. This is the same as saying that $\mathcal{P} \cup \{p'\}$ is legal, where p' is the ancestor path with tail $last(b)$ and head N_A. Thus, in order to prove (i) we must show $\mathcal{P} \cup \{p, p'\}$ is legal, provided that both $\mathcal{P} \cup \{p\}$ and $\mathcal{P} \cup \{p'\}$ are legal.

We create a total order on the nodes of T such that for each ancestor path q such that $\mathcal{P} \cup \{q\}$ is legal, all internal nodes on q precede the head of q.

Added this note:

This order is extended to an order on the paths by just comparing the heads of the paths. Since this order extends the precedes-relation, and

End note.

we derived that both $\mathcal{P} \cup \{p\}$ and $\mathcal{P} \cup \{p'\}$ are legal, according to this order $\mathcal{P} \cup \{p, p'\}$ is legal as well.

Todo: make the recursion more explicit

Insert into the total order the atom nodes in reverse order of depth (defined as the distance from the root) except for a head H of any merge path from $\mathcal{P}$. Consider the set of paths for which H is the head and the set of internal atom nodes on these paths. Insert

H into the total order as soon as all of these internal nodes are inserted. Observe that for every legal ancestor path its head node is later in the partial order than all of its internal nodes.

(ii). Apply induction on the size of $\mathcal{P}'$, thereby using (i) in the induction step and the trivial fact that $\mathcal{P} \cup \mathcal{P}'$ is legal implies that any subset of $\mathcal{P} \cup \mathcal{P}'$ is legal as well. ■

3 Hyper Tableaux with Merge Paths

Before defining the new calculus we take one more preliminary step: suppose that $B \in [\![b]\!]_{\mathcal{P}}$ for given open branch b and legal path set $\mathcal{P}$. In the trees constructed in Definition 3.1, there is a *unique* node N_B in b with $\lambda(N_B) = B$ such that N_B is visible from the leaf of b (this is proven in Lemma 3.3-(ii) below). Consequently, the ancestor merge path $ancp((b,\neg B),N_B)$ is uniquely defined, and it is denoted by $ancp((b,\neg B))$ alone.

Definition 3.1 (Hyper tableaux with merge paths)
Let T be a tree, b be a branch in T and let $L_1 \vee \cdots \vee L_n$ be a disjunction of literals. We say that T' is an *extension of T at b with* $L_1 \vee \cdots \vee L_n$ iff T' is obtained from T by attaching to the leaf of b n new successor nodes $N_1,\ldots,N_n$ that are labeled with the literals $L_1,\ldots,L_n$ in this order. For N_i ($1 \leq i \leq n$) the *clause of* N_i is defined as $clause(N_i) = L_1 \vee \cdots \vee L_n$.

A *selection function* is a total function f which maps an open tree to one of its open branches. If $f(T) = b$ we also say that b *is selected in* T *by* f.

Hyper tableaux T *for* $\mathcal{S}$ *with merge path set* $\mathcal{P}$ – or $(T,\mathcal{P})$ for short – are defined inductively as follows.

Initialization step: $(\varepsilon,\emptyset)$ is a hyper tableau for $\mathcal{S}$, where ε is a tree consisting of a root node only. Its single branch is marked as "open".

Hyper extension step with C**:** If

(i) $(T,\mathcal{P})$ is an open hyper tableau for $\mathcal{S}$ with selected branch b, and

(ii) $C = A_1,\ldots,A_m \leftarrow B_1,\ldots,B_n$ is a clause from $\mathcal{S}$ (for some $A_1,\ldots,A_m$ and $B_1,\ldots,B_n$ and $m,n \geq 0$), and

(iii) $\{B_1,\ldots,B_n\} \subseteq [\![b]\!]_{\mathcal{P}}$, and

(iv) $\{A_1,\ldots,A_m\} \cap [\![b]\!]_{\mathcal{P}} = \emptyset$ (regularity),

then $(T',\mathcal{P}')$ is a hyper tableau for $\mathcal{S}$, where

(i) T' is an extension of T at b with $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$, and

(ii) every branch $(b,\neg B_1)\ldots,(b,\neg B_n)$ of T' is labeled as closed, and

(iii) every branch $(b,A_1)\ldots,(b,A_m)$ of T' is labeled as open, and

(iv) $\mathcal{P}' = \mathcal{P} \cup \{ancp((b,\neg B_1)),\ldots,ancp((b,\neg B_n))\}$.

If conditions (i) – (iv) hold, we say that an "extension step with clause $\mathcal{A} \leftarrow \mathcal{B}$ is applicable to b".

Merge path step with p: If

(i) $(T, \mathcal{P})$ is an open hyper tableau for $\mathcal{S}$ with selected branch b, and

(ii) $p = mergep(b, b^H)$ is a non-ancestor merge path from b, for some rooted partial branch b^H of T, and

(iii) $last(b^H)$ is not the tail of a merge path in $\mathcal{P}$, and

(iv) $\mathcal{P} \cup \{p\}$ is legal,

then $(T', \mathcal{P}')$ is a hyper tableau for $\mathcal{S}$, where

(i) T' is the same as T, except that b is labeled as closed in T', and

(ii) $\mathcal{P}' = \mathcal{P} \cup \{p\}$.

If conditions (i) – (iv) hold, we say that a "merge path step with merge path p is applicable to b".

A (possibly infinite) sequence $((\varepsilon, \emptyset) = (T_0, \mathcal{P}_0)), (T_1, \mathcal{P}_1), \ldots, (T_n, \mathcal{P}_n), \ldots$ of hyper tableaux for $\mathcal{S}$ is called a *derivation*, where $(T_0, \mathcal{P}_0)$ is obtained by an initialization step, and for $i > 0$ the tableau $(T_i, \mathcal{P}_i)$ is obtained from $(T_{i-1}, \mathcal{P}_{i-1})$ by a single application of one of the other inference rules. A *derivation of* $(T_n, \mathcal{P}_n)$ is a finite derivation that ends in $(T_n, \mathcal{P}_n)$. A *refutation of* $\mathcal{S}$ is a derivation of a closed tableau. □

This definition is an extension of previous ground versions of hyper tableaux (mentioned in the introduction) by bringing in an inference rule for merge paths. Below it will be further extended to explicity handle minimal models.

The introduction of non-ancestor merge paths requires to explicitly keep track of the ancestor merge paths as well. Without non-ancestor merge paths all nodes along a branch are always visible and the calculus uses the "standard" branch semantics of setting all atoms on a (consistent) branch true. No computational overhead is introduced for checking legality. The same holds if only right hooks (or only left hooks) are used. This explains from our viewpoint the possibility of doing complement splitting [Manthey and Bry, 1988].

The purpose of the hyper extension step rule is to satisfy a clause that is not satisfied in the selected branch b. An implicit legality check for the ancestor paths added in an extension step is carried out by excluding those atoms from the branch semantics that would cause illegality when drawing an ancestor path to them. Lemma 3.3-(i) below explains more generally that every derivable hyper tableau is equipped with a legal path set.

The applicability condition (iii) in the definition of hyper extension step expresses that *all* body literals $B_1, \ldots, B_n$ are required to be present in the context b; this similarity to *hyper resolution* gave the name *hyper tableaux*.

An obvious invariant of the inference rules is that every open branch b is labeled with positive literals only. Thus $[\![b]\!]_{\mathcal{P}}$ conforms to our convention of representing interpretations as the set of atoms being true in it. On the other side, due to merge path steps there are also closed branches whose semantics consists of positive literals only.

The purpose of the merge path step inference rule is to close branches because a "proof" or a model is to be found in the branch where the drawn merge path is pointing to. In the applicability condition (iii) we insist that a non-ancestor merge path is never drawn to the tail node of another merge path. Without this condition, the clause set consisting of the single clause $A, A \leftarrow$ would admit a refutation, and this would be unsound of course (these two merge paths would form a legal set, and hence condition (iv) would not prevent us from this "refutation").

Example 3.2 (Hyper tableaux with merge paths)
(1) As a first example consider Figure 3 in Example 2.3 again. Closed branches are marked with the symbol "×" as closed. Only the tableau in the middle is constructible by the calculus, because the calculus rules forbid the derivation of a tableau with an illegal set of merge paths. In this middle tableau the left branch gets closed by a hyper extension step with the clause $\leftarrow C$, and the right branch is closed by a non-ancestor merge path step as indicated. This application of a non-ancestor merge path step corresponds to a folding-up step in model elimination [Letz *et al.*, 1994].

The right tableau shows that it would not be sound to check only non-ancestor merge paths for legality, because without ancestor merge paths legality would hold, and the tableau would be closed, although the underlying clause set is satisfiable.
(2) Figure 4 serves as an example to demonstrate the change of branch semantics as the derivation proceeds and the computation of models.

Suppose that the hyper tableau $\boxed{1}$ has been constructed. The semantics of the right branch $b = (B,E,C)$ is $[\![b]\!]_{\mathcal{P}} = \{B,E,C\}$. Suppose that this branch can be extended further. Suppose that the left subtree contains an open branch $b_{\ldots}$ that makes A and C true. This is indicated by the set $[\![b_{\ldots}]\!]_{\mathcal{P}} = \{A,C,\ldots\}$. Further suppose that this is a minimal model.

Next, let a merge path step be applied with non-ancestor merge path p to the tableau $\boxed{1}$, yielding the tableau $\boxed{2}$. By this step, b is closed and hence its interpretation is rejected for the time being. A second effect of this step is that the node labeled with $\boxed{A}$ becomes invisible in $b_{\ldots}$. Thus $[\![b_{\ldots}]\!]_{\mathcal{P}\cup\{p\}} = \{C,\ldots\}$. Now, this new interpretation has to be "repaired" by bringing in A again. This is done in the next step by extending with $A \vee B$ yielding a tableau $\boxed{3}$ (which is not depicted). Notice that the minimal model $[\![b_{\ldots}]\!]_{\mathcal{P}}$ is indeed reconstructed, only in a different order. In order to reconstruct the rejected interpretation $\{B,E,C\}$ from above that was rejected by the merge path step, a hyper extension step below the new B node with E is carried out. This leads to the tableaux $\boxed{4}$. Notice that the new branch with semantics $\{C,\ldots,B,E\}$ possibly contains more elements than the corresponding one with semantics $\{B,E,C\}$ before.

In other words, the *search* of all these extra elements, which would have been carried out below b when the merge path step would not have been carried out, has been replaced by the two hyper extension steps leading from $\boxed{2}$ to $\boxed{4}$. Of course, this in-

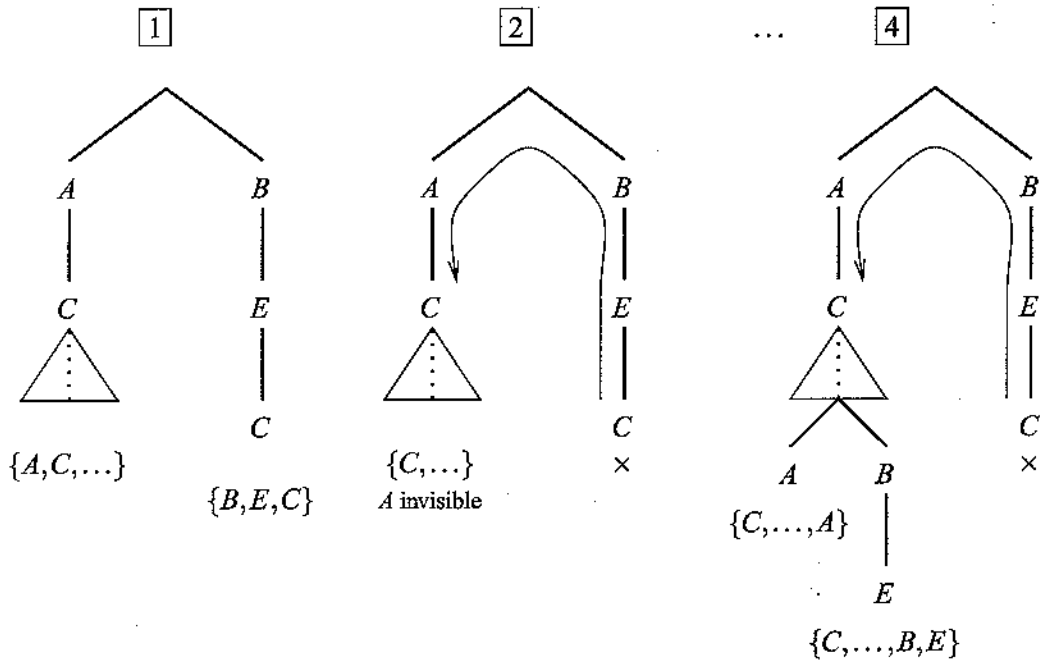

Figure 4: A snapshot from a hyper tableau derivation with merge paths. The sets represent the interpretations extracted from the branches written above them.

volves search as well, *but it can be much shorter*. Indeed, from a practical point of view this observation is the main point of the whole approach. The diagnosis application above in Section 1.2 was mentioned in support of this argument.

It is worth emphasizing that the re-computation of models happens only in the case of non-ancestor merge paths with their head in *open* branches. Merge paths into *closed* branches are "cheap" in that no re-computation is necessary. Thus, in a sense, refutational theorem proving, which would stop with failure after the first open finished branch (cf. Def. 4.3 below) is found, is "simpler" than computing models. □

Some more comments on the inference rules: in the applicability condition (iii) of the merge path step we insist that a non-ancestor merge path is never drawn to the tail node of another merge path. This is needed to guarantee *soundness*. Without this condition, the clause set consisting of the single clause $A, A \leftarrow$ would admit a refutation by drawing two such non-ancestor merge paths from one A-node to the other. Since these two non-ancestor merge paths are not related in the $\prec$-relation, the resulting path set would be legal. Thus condition (iv) would not prevent us from this "refutation".

Obviously, it would be useful to "reuse" the leaf A of a branch that was closed by a merge path step with some merge path p. This can be done within the bounds of legality by choosing a merge path to the head of p – which is also labeled with A – instead to the tail of p.

Lemma 3.3 (Basic invariants of inference rule applications)
For every hyper tableau $(T, \mathcal{P})$ the following holds:

(i) $\mathcal{P}$ *is legal.*

(ii) For every $A \in [\![b]\!]_{\mathcal{P}}$ *for given open branch* b *of* T*, there is a* unique *node* N_A *in* b *with* $\lambda(N_A) = A$ *and such that* N_A *is visible from the leaf* $last(b)$ *of* b*.*

Item (i) is relevant for the soundness of the overall calculus; item (ii) proves the existence of the *ancp* function as used in the definition of hyper extension step.

Proof. Suppose that $(T, \mathcal{P})$ is constructed by the derivation $(T_0, \mathcal{P}_0), \ldots, ((T_m, \mathcal{P}_m) = (T, \mathcal{P}))$. For both items the proof is by induction on m.

(i). The base case with $m = 0$ is trivial. Hence we turn to the induction step, let $m > 0$ and assume the result to hold for derivations of length $< m$.

If $(T_m, \mathcal{P}_m)$ is obtained by a merge path step, then legality of $\mathcal{P}_m$ is part of the applicability condition of the merge path step. If $(T_m, \mathcal{P}_m)$ is obtained by a hyper extension step with clause $\mathcal{A} \leftarrow B_1, \ldots, B_n$, we think of $\mathcal{P}_m$ as given by the sequence

$$\begin{aligned} \mathcal{P}^0 &= \mathcal{P}_{m-1} \\ \mathcal{P}^1 &= \mathcal{P}^0 \cup \{ancp((b, \neg B_1))\} \\ &\vdots \\ \mathcal{P}_m = \mathcal{P}^n &= \mathcal{P}^{n-1} \cup \{ancp((b, \neg B_n))\} \end{aligned}$$

By induction on n we show that $\mathcal{P}^n$ is legal and that $[\![b]\!]_{\mathcal{P}^n} = [\![b]\!]_{\mathcal{P}^0}$. The case with $n = 0$ being trivial ($\mathcal{P}^0 = \mathcal{P}_{m-1}$ is legal by the outer induction hypothesis) we turn immediately to the induction step. Hence suppose that $n > 0$ and that the result holds for $n - 1$. $B_n \in [\![b]\!]_{\mathcal{P}^0}$ holds by definition of hyper extension step. By the induction hypothesis thus also $B_n \in [\![b]\!]_{\mathcal{P}^{n-1}}$. Equivalently, the node N_{B_n} in b labeled with B_n is visible from $last(b)$. It is straightforward to show that N_{B_n} is visible from the new node labeled with $\neg B_n$ as well. Hence $\mathcal{P}^{n-1} \cup \{ancp((b, \neg B_n))\} == \mathcal{P}^n$ is legal as well.

Lemma 2.5-(i) immediately gives us $[\![b]\!]_{\mathcal{P}^n} = [\![b]\!]_{\mathcal{P}^{n-1}}$. By the induction hypothesis $[\![b]\!]_{\mathcal{P}^{n-1}} = [\![b]\!]_{\mathcal{P}_0}$. Together thus $[\![b]\!]_{\mathcal{P}^n} = [\![b]\!]_{\mathcal{P}^0}$ as claimed.

(ii). The base case with $m = 0$ is trivial. Hence we turn to the induction step, let $m > 0$ and assume the result to hold for derivations of length $< m$.

It suffices to consider hyper extension steps only, and thereby consider the extended branch b only, because in all other cases the interpretation of an open branch either remains the same or is becoming smaller. Consequently, since T_{m-1} does not contain multiple occurrences of A in any one of these branches by the induction hypothesis, this also holds for T_m.

The single critical case is that a hyper extension step was applied to the open branch b_{m-1} in T_{m-1} with clause $\mathcal{A} \leftarrow \mathcal{B}$. Suppose, to the contrary, that the open branch $b = (b_{m-1}, A)$ in T_m contains two nodes N_1 and N_2 both labeled with A for some $A \in \mathcal{A}$ and

both are visible from $last((b_{m-1},A))$ in $\mathcal{P}_m$. By the induction hypothesis, property (ii) holds for b_{m-1} and $\mathcal{P}_{m-1}$. Hence, one of these two nodes, say N_1 must be the new leaf $last((b_{m-1},A))$, and the other one – N_2 – must be some node from b_{m-1}. Since N_2 is visible from $last((b_{m-1},A))$ in $\mathcal{P}_m$, it is visible from $last(b_{m-1})$ in $\mathcal{P}_{m-1}$ as well (because $\mathcal{P}_{m-1} \subseteq \mathcal{P}_m$). In other words, $A \in [\![b_{m-1}]\!]_{\mathcal{P}_{m-1}}$. But then, the considered hyper extension step is not applicable because condition (iv) in the definition of hyper extension step is violated. Contradiction. ■

By simply leaving out merge path steps in derivations one recognizes that the new calculus is a generalization of previous ground versions of the hyper tableaux calculus. The branch semantics of previous versions was to collect the positive literals along an open branch. This coincides with the new calculus then, because without non-ancestor merge paths all predecessor nodes of a leaf are visible, and hence true. Furthermore, due to the absence of a legality test in hyper extension steps no computational overhead is introduced.

4 Finite Derivations

As expected, we are interested in a complete calculus. This requires to introduce a notion of *fairness*. Whenever one has to cope with infinite derivations, defining fairness requires some technicalities that would seem like an overkill for the propositional case which is "expected" to terminate (see [Baumgartner *et al.*, 1996; Baumgartner, 1998; Baumgartner *et al.*, 1999] for first-order versions of related calculi).

Unfortunately, our calculus does not terminate in general, i.e. there are infinite derivations (for finite clause sets), although we employ the "regularity" test (cf. Def. 3.1). This is due to deep merge paths – without them, termination is straightforward to prove.

Example 4.1 (Nontermination) Consider the satisfiable clause set $\{(A,B \leftarrow),(B,C \leftarrow),(A,D \leftarrow),(C \leftarrow A)\}$. Its minimal models are $\mathcal{I}_1 = \{A,C\}$ and $\mathcal{I}_2 = \{B,D\}$.

Figure 5 shows steps of a non-termination derivation. Tableau $\boxed{1}$ is obtained by two hyper extension steps and a merge path step as drawn. No further hyper extension step is applicable to the branch (A,C). Suppose that branch (B) is selected next. Two further hyper extension steps yield the tableau $\boxed{2}$. To the branch (B,A,C) no more hyper extension step is applicable[5]. We apply a merge path step pointing into the branch (A,C). Since in the target branch the atom A becomes invisible, further extension on (A,C) is possible. This is done and the result is tableau $\boxed{3}$. In this and the subsequent tableaux, "old" merge paths and closed subtrees are only indicated using dots. Tableau $\boxed{4}$ is obtained from tableau $\boxed{3}$ by a merge path step as indicated and subsequent extension with $A,B \leftarrow$. Similarly, tableau $\boxed{5}$ is obtained by a merge path step and a hyper extension step with $B,C \leftarrow$. A final application of a merge path step to tableau $\boxed{5}$ yields tableau $\boxed{6}$.

[5]This example thus also demonstrates the – well-known – fact that hooks and related techniques such as complement splitting are too weak to guarantee minimal model soundness.

Now, a circular situation is obtained: tableau 6 contains two open branches. The one, $b_1 = (A,C,A,C)$ in the left subtree has the branch semantics $\{A,C\}$, and the other, $b_2 = (B,D,B)$ in the right subtree has the branch semantics $\{B\}$. This constitutes a loop, because tableau 1 has the same semantics. □

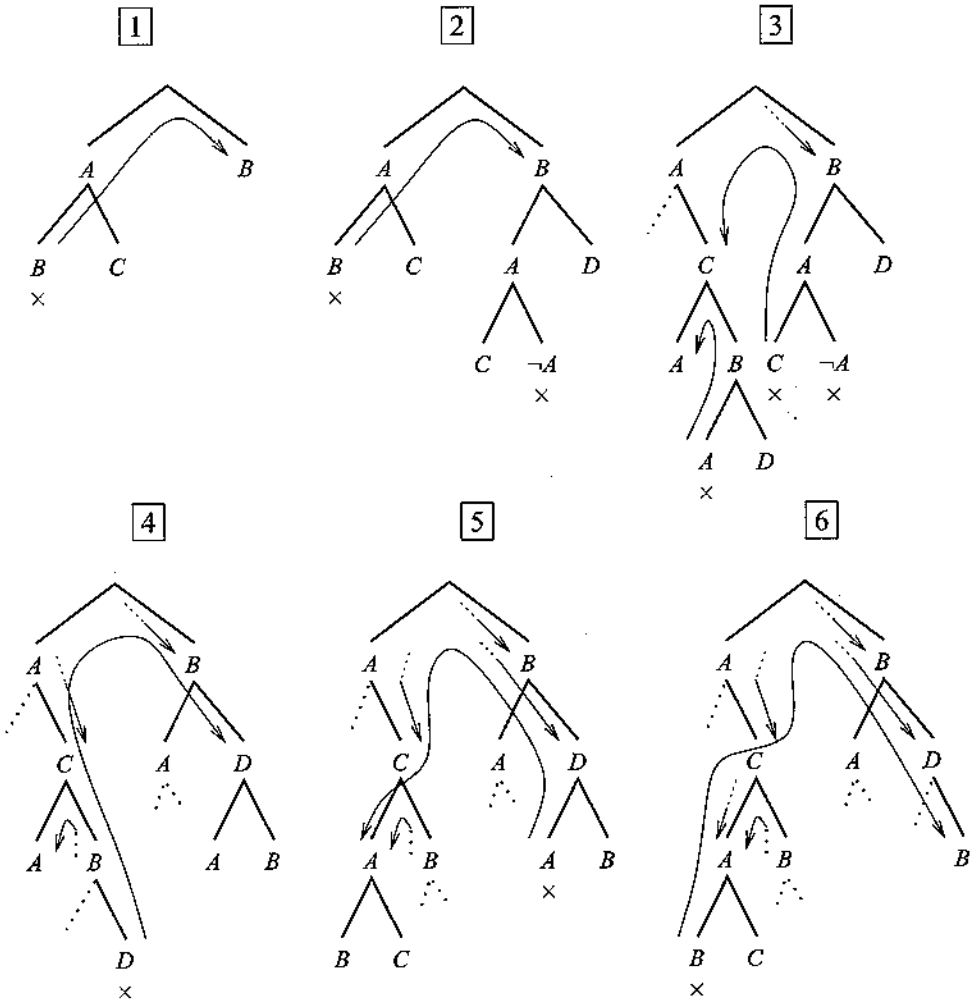

Figure 5: Steps from a non-termination hyper hyper tableau derivation with merge paths.

The previous example demonstrates that merge paths have to be applied with care in order to get a terminating calculus.

As a consequence we propose the following technique:

Theorem 4.2 (Sufficient Termination Criteria)
Every derivation $(T_0,\mathcal{P}_0),\ldots,(T_n,\mathcal{P}_n),\ldots$ *satisfying one of the following criteria is finite.*

(i) *For every $(T_i, \mathcal{P}_i)$, where $i \geq 0$, an applicable merge path step with merge path p is not carried out if for some open branch b in T_i more than an a priori fixed number max of occurences of some label A is invisible from $last(b)$ in $\mathcal{P}_i \cup \{p\}$.*

(ii) *For every $(T_i, \mathcal{P}_i)$, where $i \geq 0$, an applicable merge path step with merge path p is not carried out if $\mathcal{P}_i \cup \{p\}$ contains a chain $p_1 \prec \cdots \prec p_m$ of non-ancestor merge paths, where m is greater than an a priori fixed number max.*

I'm not so sure about (ii) any more: taking only merge path steps into consideration might be too weak – they could be connected by ancestor paths in alteration, therefore admitting chains of unbounded length. On the other side, taking ancestor paths into consideration would prevent us from some hyper extension steps. That cannot be allowed. I'm not sure whether Joe's proof solves the problem.

Criterion (1) avoids infinite derivations by bounding repetitions of the same literal along branches. A trivial instance is $max = 0$. Then no deep merge paths but only hooks are possible. Observe that the derivation in Example 4.1 still conforms to the limit $max = 1$.

The idea underlying Criterion (1) is that one should not too often be forced to repeat the derivation of an atom that becomes repeatedly invisible on a branch.

Criterion (2) avoids infinite derivations by monitoring the length of chains. Clearly, when attempting to insert a non-ancestor merge path p into $\mathcal{P}$ it suffices to check for those non-ancestor merge paths in $\mathcal{P}$ that form a chain wrt. $\prec$ containing p.

Observe that both criteria are made such that they never prevent a hyper extension step from being carried out. This is important to guarantee that fair derivations still exist (cf. Definition 4.3). Essentially, this implies together with the termination property that the completeness results hold.

Proof. (i). With the criteria, the length of branches is bound to be less or equal to $(max+1) \cdot |\Sigma|$. For, suppose that some atom A occurs $m > max+1$ times in some branch b in some tableau of the derivation. Consider the point in time i where the m-th (for some arbitrary $m > max+1$) occurrence of A was introduced to b by a hyper extension step to a prefix (i.e. rooted partial branch) b_i of b. Let $(T_i, \mathcal{P}_i)$ be that tableau containing b_i. By the applicability condition (iv) of hyper extension step, no node labeled with A in b_i can be visible from $last(b_i)$. Since we consider the case that A was introduced the m-th time to b, it must be that $m-1 > max$ occurences of A occur in b_i, and each of them is invisible from $last(b_i)$ in $\mathcal{P}_i$. Since hyper extension steps do not affect visibility (cf. Lemma 2.5), there must have occured in the derivation a merge path step that caused what was just concluded. But then the derivation would not conform to (i).

(ii).

?

■

Due to this criterion we consider from now on only finite derivations.

Definition 4.3 (Redundancy, Fairness)
Suppose as given some hyper tableau $(T, \mathcal{P})$ for $\mathcal{S}$. A clause $\mathcal{A} \leftarrow \mathcal{B}$ is called *redundant* in an open branch b of T wrt. $\mathcal{P}$ iff $[\![b]\!]_{\mathcal{P}} \models \mathcal{A} \leftarrow \mathcal{B}$ (iff $\mathcal{B} \subseteq [\![b]\!]_{\mathcal{P}}$ implies $\mathcal{A} \cap [\![b]\!]_{\mathcal{P}} \neq \emptyset$).

A branch b of T is called *finished wrt.* $\mathcal{P}$ iff

(i) b is closed, or else

(ii) every clause $\mathcal{A} \leftarrow \mathcal{B} \in \mathcal{S}$ is redundant in b wrt. $\mathcal{P}$.

The term *unfinished* means "not finished".

Now suppose as given a finite derivation $D = (T_0, \mathcal{P}_0), \ldots, (T_n, \mathcal{P}_n)$ from $\mathcal{S}$ with selection function f. D is called *fair* iff

(i) D is a refutation, i.e. T_n is closed, or else

(ii) $f(T_n)$ is finished wrt. $\mathcal{P}_n$.

The selection function f is called a *model computation selection function* iff f maps a given open hyper tableau $(T, \mathcal{P})$ to an unfinished branch wrt. $\mathcal{P}$, provided one exists, else f maps T to some other open (finished) branch.

Quite often we omit the term "wrt. $\mathcal{P}$" and let $\mathcal{P}$ be given by the context. □

The *existence* of fair derivations is straightforward because we insist on *finite* derivations. Notice that any input clause not redundant so far in a branch b can be made redundant by simply carrying out an extension step with.

The idea behind a *model-computation selection function* is that no derivation should stop with an unfinished branch. Since finished open branches constitute models, with such a selection function *every* model will be computed. The next section makes this more precise.

5 Soundness and Minimal Model Completeness

First we prove soundness. Instead of a refutational soundness formulation we use the contrapositive formulation and in a stronger form. It is stronger than a refutational formulation, because it not only says that a satisfiable clause set $\mathcal{S}$ admits no refutation, but also that any minimal model of $\mathcal{S}$ is actually computed (in a sense). This in turn is needed to prove minimal model completeness.

This is the main technical lemma:

Lemma 5.1 (Soundness lemma)
Let $(T, \mathcal{P})$ be a hyper tableau for satisfiable clause set $\mathcal{S}$. Then for every minimal model $\mathcal{I}$ of $\mathcal{S}$ there is an open branch b of T such that $[\![b]\!]_{\mathcal{P}} \subseteq \mathcal{I}$.

5.1 Proof of Lemma 5.1 – Simulation by "atomic cuts"

Let us first deal with the trivial case that T consists of a root node alone. There is only one branch in T, namely (N_0), it is open and $[\![(N_0)]\!]_{\mathcal{P}} = \emptyset$ proves the claim. Henceforth assume that T does not consist of a root node alone.

We extend the calculus by the following inference rule:

Atomic cut: Let T be an open hyper tableau with merge path set $\mathcal{P}$ for $\mathcal{S}$ with selected open branch b. If $A \in \Sigma$ is an atom, then the tree T' is a hyper tableau for $\mathcal{S}$ with merge paths set $\mathcal{P}$, where T' is an extension of T at b with $\neg A \vee A$[6].

Now define *hyper tableau with cut* like hyper tableau with merge paths as in Definition 3.1, but extended by the new inference rule. The tableaux of the old calculus are called *hyper tableau without cut*. Atomic cuts have also been used in [Letz *et al.*, 1994] to prove the soundness of the folding up/folding down inference rules in model elimination.

Notice that now branches contain negative literals in other as leaf positions as well, and branches containing negative literals may well be open. We do *not* introduce a *closure rule*, according to which a branch b is marked as closed if its semantics contains two complementary literals. This would complicate matters unnecessary.

Instead we temporarily define a branch $b = (N_0, N_1, \ldots, N_n)$ as *inconsistent* iff $\{A, \neg A\} \subseteq \{\lambda(N_1), \ldots, \lambda(N_n)\}$ for some atom A; *consistent* means "not inconsistent". This definition is intentionally *not* based on visibility (cf. Def. 2.4)

In order to come to a semantics definition comparable to the calculus without the atomic cut rule, we simply forget about the negative literals. More formally, define

$$[\![b]\!]^+_{\mathcal{P}} = \{A \in [\![b]\!]_{\mathcal{P}} \mid A \text{ is a positive literal}\}$$

for any consistent and open branch b in a hyper tableau with cut[7].

The plan now is the following: we show that all non-ancestor merge paths can be transformed away in a semantical preserving way by using atomic cuts. The claimed property in the lemma statement is proven for the resulting tableau and translated back for the original tableau T using an invariant of the transformation.

We work with this mapping to hyper tableau with cut, because (a) the cut rule is a standard tool for proving soundness of refinements like the ones we consider, (b) it might be interesting in its own right how cuts can simulate our non-ancestor merge paths (cf. also Section 5.2 below).

We approach the proof top-down. The transformation t to be defined below takes a hyper tableau with cut $(T, \mathcal{P})$ and returns a hyper tableau with cut $t(T, \mathcal{P}) = (T', \mathcal{P}')$. The transformation t satisfies the following property:

Invariant of t: for every consistent and open branch b' of $(T', \mathcal{P}')$ there is a consistent and open branch b of $(T, \mathcal{P})$ such that

$$[\![b]\!]^+_{\mathcal{P}} \subseteq [\![b']\!]^+_{\mathcal{P}'} \ . \tag{1}$$

The transformation itself is defined, provided that $\mathcal{P}$ contains at least one non-ancestor merge path; it leaves us with $\mathcal{P}'$ containing exactly one non-ancestor merge path less.

[6]Notice that we do not have a restriction such as $A \notin [\![b]\!]_{\text{cut}}$ that would enforce that a cut with the same atom is carried out only once along a branch; indeed, we could not afford such a restriction, because the transformation below would violate it in general.

[7]This definition could thus be also applied to branches closed in a merge path step, but it does not play no role.

Hence, when applied to the initially given tableau $(T, \mathcal{P})$ (without any cuts) it will leave us after finitely many, say, n steps with a hyper tableau with cut $(T_{\text{cut}}, \mathcal{P}_{\text{cut}})$ where $\mathcal{P}_{\text{cut}}$ contains no single non-ancestor merge path. Since no non-ancestor merge paths are present, we can forget wrt. branch semantics about the ancestor merge paths and safely set $\mathcal{P}_{\text{cut}} = \emptyset$. More formally, it holds $[\![b_{\text{cut}}]\!]_{\emptyset} = [\![b_{\text{cut}}]\!]_{\mathcal{P}_{\text{cut}}}$ for any branch b_{cut} of $(T_{\text{cut}}, \mathcal{P}_{\text{cut}})$. The technical justification for this step is given by Lemma 2.5-(ii).

Below we show that for some consistent and open branch b_{cut} of $(T_{\text{cut}}, \mathcal{P}_{\text{cut}})$ it holds $[\![b_{\text{cut}}]\!]^+_{\emptyset} \subseteq \mathcal{I}$. Using (1) n times then gives us the existence of a consistent and open branch b of $(T, \mathcal{P})$ such that $[\![b]\!]^+_{\mathcal{P}} \subseteq [\![b_{\text{cut}}]\!]^+_{\emptyset}$. Now, since $(T, \mathcal{P})$ is a hyper tableau without cut, the property of b being open and consistent means that $[\![b]\!]_{\mathcal{P}}$ consists of positive literals only. In other words, $[\![b]\!]_{\mathcal{P}} = [\![b]\!]^+_{\mathcal{P}}$. Now putting the just obtained results together we get $[\![b]\!]_{\mathcal{P}} \subseteq \mathcal{I}$ as claimed in the lemma statement. This completes the first step of the proof.

The next subgoal to be shown is the following:

Existence of a consistent and open branch b_{cut} of $(T_{\text{cut}}, \mathcal{P}_{\text{cut}})$ with $[\![b_{\text{cut}}]\!]^+_{\emptyset} \subseteq \mathcal{I}$: In this subproof we can write instead of "consistent and open" simply "consistent", because without the presence of non-ancestor merge paths "consistent" implies "open". T_{cut} contains at least one consistent branch b_{cut} such that the following holds

$$\neg A \notin [\![b_{\text{cut}}]\!]_{\emptyset} \quad \text{for every } A \in \mathcal{I}, \text{ and} \tag{2}$$

$$A \notin [\![b_{\text{cut}}]\!]_{\emptyset} \quad \text{for every } \neg A \in \neg\overline{\mathcal{I}}, \text{ where } \neg\overline{\mathcal{I}} = \{\neg A \mid A \in \Sigma \setminus \mathcal{I}\} \tag{3}$$

That is, $[\![b_{\text{cut}}]\!]_{\emptyset}$ contains no contradiction to $\mathcal{I}$. Notice that $\mathcal{I}$ is a model for $\neg\overline{\mathcal{I}}$ by convention of representing interpretations.

Reason for the existence of such a b_{cut}: suppose, to the contrary, that for every consistent branch b_{cut} there is a $A \in \mathcal{I}$ such that $\neg A \in [\![b_{\text{cut}}]\!]_{\emptyset}$ or there is a $\neg A \in \neg\overline{\mathcal{I}}$ such that $A \in [\![b_{\text{cut}}]\!]_{\emptyset}$. This means that every consistent branch would be inconsistent after extending with some positive unit clause $A \in \mathcal{I}$ or some negative unit clause $\neg A \in \neg\overline{\mathcal{I}}$. Let T'_{cut} be that tableau. T'_{cut} is a "standard" tableau with cut (cf. e.g. [Fitting, 1990]) for the input clause set $\mathcal{S} \cup \mathcal{I} \cup \neg\overline{\mathcal{I}}$. Hence, by the well-known soundness result for such tableau[8], $\mathcal{S} \cup \mathcal{I} \cup \neg\overline{\mathcal{I}}$ must be unsatisfiable. This, however, immediately yields a contradiction to the given assumption that $\mathcal{I}$ is a model for $\mathcal{S}$. Hence T_{cut} contains at least one consistent branch b_{cut} with the claimed properties (2) and (3).

Now let b_{cut} be any such branch. Next we *claim* that $[\![b_{\text{cut}}]\!]_{\emptyset} \subseteq \mathcal{I} \cup \neg\overline{\mathcal{I}}$. Let b'_{cut} be the rooted partial branch of b_{cut} of maximal length which is in order with $\mathcal{I} \cup \neg\overline{\mathcal{I}}$. That is, if $b_{\text{cut}} = (N_0, N_1, \ldots, N_n)$ then $b'_{\text{cut}} = (N_0, N_1, \ldots, N_m)$ for some maximal $m \leq n$ and such that and $[\![b'_{\text{cut}}]\!]_{\emptyset} \subseteq \mathcal{I} \cup \neg\overline{\mathcal{I}}$ (trivially b'_{cut} is consistent as well, because we have no non-ancestor merge paths).

It is clear that $n > 0$, because T was assumed to be different from the tree consisting of a root node alone, and the transformation t does not produce this situation.

Now, either $m = n$ and the claim holds, or else $m < n$. In the latter case b'_{cut} exists indeed, since with $m = 0$ we have $[\![(N_0)]\!]_{\emptyset} = \emptyset \subseteq \mathcal{I} \cup \neg\overline{\mathcal{I}}$ and the condition is satisfied ($m = 0$ it might be not maximal, though). Thus let $m < n$ be maximal now. This case leads to a contradiction.

[8] What we call "inconsistent" here is often called "closed" in the literature.

Since $m < n$ there is at least one successor node N_{m+1} of N_m in b'_{cut}, and by maximality we know that $\lambda(N_{m+1}) \notin \mathcal{I} \cup \neg\bar{\mathcal{I}}$ although $\lambda(N_{m+1}) \in [\![b_{\text{cut}}]\!]_{\emptyset}$.

Now, if $\lambda(N_{m+1}) = A$ is a positive atom then $A \notin \mathcal{I} \cup \neg\bar{\mathcal{I}}$ is the same as $A \notin \mathcal{I}$ which implies $\neg A \in \neg\bar{\mathcal{I}}$ by definition of $\neg\bar{\mathcal{I}}$. But then, by (3) we conclude that $A \notin [\![b_{\text{cut}}]\!]_{\emptyset}$, which is a plain contradiction to what was just concluded by the maximality assumption. The case that $\lambda(N_{m+1}) = \neg A$ labels a negative atom yields a contradiction by symmetrical arguments.

Hence, the assumption $m < n$ must have been wrong. Therefore $m = n$ and the *claim* above holds. With respect to positive atoms the claim can be slightly rewritten as $[\![b_{\text{cut}}]\!]^+_{\emptyset} \subseteq \mathcal{I}$, which is nothing but the claim of the current subgoal.

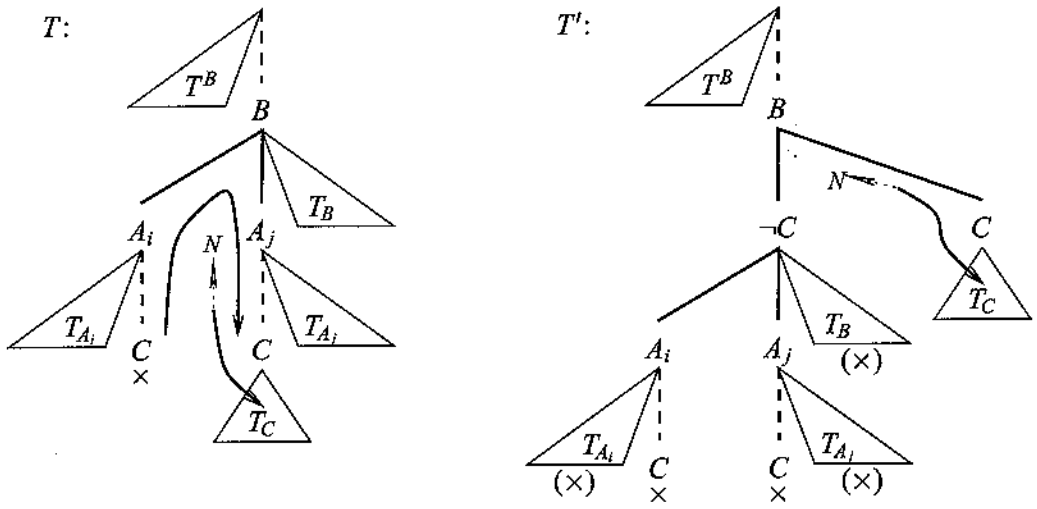

Figure 6: The transformation t

The transformation t: The left side in Figure 6 displays the most general situation. As said, we assume that $\mathcal{P}$ contains at least one non-ancestor merge path p. It is the one drawn in the depicted tree and that is transformed away by t.

Figure 6 is to be read as follows: dashed lines mean partial branches. For instance, the top leftmost dashed line leading to B means the partial branch p_B from the root to the node (inclusive) labeled with B. Triangles are certain forests. The top leftmost triangle T^B means the forest that is obtained from T by deleted the branch p_B and the whole subtree below the node B. The most appropriate intuition is to think of trees as branch sets. Then the triangle T^B is simply the set of the branches obtained from T by deleting all branches that contain p_B.

The solid lines, just like the ones below B indicate a hyper extension step; here, it is supposed that a hyper extension step with clause $C = A_1, \ldots, A_n \leftarrow B$ has been carried out to node p_B, and all the literals short of A_i and A_j (for some $i, j \in \{1, \ldots, n\}$ and

$i \neq j$) are attached to nodes in the forest T_B. The assumed non-ancestor merge path p is indicated with tail node C (left) and head node C (right) and turn point B.

There might be other non-ancestor merge paths in $\mathcal{P}$, in particular some where the head of p is an inner node of. This possibility is indicated in Figure 6 as well, by the arrow pointing into T_C. The tail of this non-ancestor merge path, say p_C is a leaf node N somewhere in T.

Inconsistent or closed branches are marked by "$\times$".

The effect of the transformation t is shown on the right. The non-ancestor merge path p is removed. Instead a cut with C – the label of the head and tail of p – is inserted at p_B, i.e. the turn point of p. The subtree below T_C (excluding C) is moved to the node labeled with C that was introduced by the atomic cut step. These operations are understood to move merge paths as well. For example, the non-ancestor merge path p_C still has the same head and tail node, but they are possibly located in different places in T' now, and also a different turn point might result.

Several properties have to be argued for:

Initially, legality of $\mathcal{P}$ holds by Lemma 3.3-(i). Since $\mathcal{P}' \subset \mathcal{P}$, $\mathcal{P}'$ is trivially legal as well. In other words, legality is preserved by t.

Since $\mathcal{P}$ is legal, the subtableau T_C in T cannot depend from any node in $(B, A_j, \ldots, C)$, because an ancestor merge path to any of these nodes would violate the given legality of $\mathcal{P}$ due to the presence of p. More precisely, all branches closed via ancestor merge paths in T_C during hyper extension steps remain closed after application of t (moving T_C below the C node from the atomic cut step), because none of those branches was closed by an ancestor merge path to an atom in $(B, A_j, \ldots, C)$. This proves that the hyper extension steps in T_C after transformation indeed exist. Hence $(T', \mathcal{P}')$ is a hyper tableau with cut.

Notice that the branch $(\ldots, B, \neg C, A_j, \ldots, C)$ is inconsistent as indicated. It is important that this branch becomes inconsistent, since the **Invariant** requires that there is no open and consistent branch (after the transformation) that does not have a counterpart before the transformation[9]

The set of ancestor nodes visible from a node in T_C or any other node below B is possibly *increased* by the transformation, because the removal of p can render nodes in the branch $(\ldots, B)$ visible that were formerly invisible due to a merge path with the head in $(A_i, \ldots, C)$ (C excluded). This is explains the "$\subseteq$" instead of a possibly expected "$=$" in the **Invariant**.

The critical part of the construction is the move of T_C. Observe that all the nodes in $(A_j, \ldots, C)$ in $(T, \mathcal{P})$ are invisible from all nodes in T_C due to the presence of p. Hence, the invariant also holds for branches the branches going into T_C.

After transformation some branches might become inconsistent due to the presence of $\neg C$. This is indicated by "$(\times)$"; this causes no problems, as the **Invariant** does not require preservation of open branches in the forward direction of the transformation.

[9] It might be that $\neg C$ is not visible from C due to some non-ancestor merge path, e.g. one containing B with the same head node C as p. This motivates our definition above not to take visibility into account in the definition of "inconsistent branch".

In sum, we defined a transformation t that satisfies the properties needed to complete the proof of Lemma 5.1. Hence the proof of Lemma 5.1 is completed ■

Some concluding remarks.
(i) What if $\mathcal{P}$ would not be legal? Then, there would be no guarantee that all the branches in T_C closed during hyper extension steps remain closed by the transformation. If the transformation would nethertheless be applied (with possibly some modifications) to an unsoundly closed tableau T, then an *open* tableau T_{cut} would result, because the atomic cut rule is sound.
(ii) Since $\mathcal{P}$ is legal it is extendible to a partial order $\ll$. Let p be any minimal element. This implies that there is no non-ancestor merge path $p' \in \mathcal{P}$ such that the head of p' is an inner node of p. When modifying the transformation t such that it is applied to such a minimal p, then 1 in the **Invariant** can be strengthened towards $[\![b]\!]^+_{\mathcal{P}} = [\![b']\!]^+_{\mathcal{P}'}$ because using a minimal p for the transformation guarantees that the visibility of ancestor nodes of the turnpoint B remains the same after removing p. With this strengthened invariant the result would be provable as well, but we prefer the *free* choice of the next non-ancestor merge path to be transformed away. This might turn out advantageous for implementations. They could mimic non-ancestor merge paths by atomic cuts in the *temporal* order they are constructed.

Theorem 5.2 (Refutational Soundness) *Let D be a refutation of $\mathcal{S}$. Then $\mathcal{S}$ is unsatisfiable.*

Proof. Sketch: basically, use the soundness results in [Horton and Spencer, 1997a] on clause trees. To do so, write down a closed hyper tableau T with merge paths $\mathcal{P}$ as a clause tree refutation with merge paths. Thereby use phantom edges, and model the simultaneous operation on all the body literals in hyper extension steps by a sequence of ancestor merge paths. T is thus transformed into a clause tree refutation. In this mapping the legality of merge paths $\mathcal{P}$ (cf. Lemma 3.3) carries over to the clause tree refutation. Now use the soundness result for clause trees with merge paths. It suffices to sketch this possibility, because there is an alternative proof from scratch:

Alternative proof: We prove the contrapositive direction. So let $\mathcal{S}$ be satisfiable, and let $\mathcal{I}$ be a minimal model. By Lemma 5.1 then every hyper tableau for $\mathcal{S}$ contains an open branch. Hence no refutation of $\mathcal{S}$ exists, which was to be shown. ■

The interest in the alternative proof, and in particular Lemma 5.1 comes from the application of Lemma 5.1 in proving minimal model *completeness*. For this we need the stronger claim of Lemma 5.1, but not just the non-existence of refutations for satisfiable clause sets.

Theorem 5.3 (Minimal Model and Refutational Completeness) *Let f be a model computation selection function and D be a finite, fair derivation from $\mathcal{S}$ of the hyper tableau $(T, \mathcal{P})$. Then, for every minimal model $\mathcal{I}$ of $\mathcal{S}$ there is an open branch b in T such that $[\![b]\!]_{\mathcal{P}} = \mathcal{I}$ (minimal model completeness).*

If no minimal model $\mathfrak{I}$ of $\mathcal{S}$ exists (that is, $\mathcal{S}$ is unsatisfiable) then D is a refutation, i.e. T is closed (refutational completeness).

Proof. First suppose that $\mathcal{S}$ is satisfiable, and let $\mathfrak{I}$ be any minimal model of $\mathcal{S}$. By the soundness lemma (Lemma 5.1) we know that T contains an open branch b with $[\![b]\!]_{\mathcal{P}} \subseteq \mathfrak{I}$. For this particular b even $[\![b]\!]_{\mathcal{P}} = \mathfrak{I}$ holds. For, suppose to the contrary, that $[\![b]\!]_{\mathcal{P}} \neq \mathfrak{I}$ holds, i.e. $[\![b]\!]_{\mathcal{P}} \subset \mathfrak{I}$. We are given that D is fair. Since f is a model computation selection function, this means that b is finished. This in turn means that every clause from $\mathcal{S}$ is redundant in b. In other words, $[\![b]\!]_{\mathcal{P}}$ is a model for $\mathcal{S}$. Since $[\![b]\!]_{\mathcal{P}} \subset \mathfrak{I}$ we have a contradiction to the given minimality of $\mathfrak{I}$. Hence the assumption that $[\![b]\!]_{\mathcal{P}} \neq \mathfrak{I}$ must be wrong, and thus the first part of the theorem holds.

Now, suppose that $\mathcal{S}$ is unsatisfiable, but that $\mathcal{T}$ is not closed. Let b be any open branch of T. Since $\mathcal{S}$ is unsatisfiable, $\mathfrak{I} = [\![b]\!]_{\mathcal{P}}$ must falsify some clause $\mathcal{A} \leftarrow \mathcal{B}$ from $\mathcal{S}$. In other words, $\mathcal{B} \subseteq \mathfrak{I}$ and $\mathcal{A} \cap \mathfrak{I} = \emptyset$. This shows that a hyper extension step is applicable to b with clause $\mathcal{A} \leftarrow \mathcal{B}$ (cf. applicability conditions (iii) and (iv) in the defintion of hyper extension step). Hence b is not finished and thus D is not fair. But D was given as fair. Contradiction. ■

Notice that our definition of fairness does not insist on carrying out merge path steps. Consequently, the minimal model completeness result holds whether merge paths are applied or not. In other words, merge path steps are completely optional. Indeed, as shown above, insisting on certain merge paths can easily cause nontermination.

5.2 Why Merge Paths and not "Atomic Cuts"

In the proof of Lemma 5.1 we indicated how atomic cuts can be used to simulate non-ancestor merge paths. So, the question might arise why not directly use these cuts. The answer is manifold. First, by the mere fact that the simulation exists we get insights how merge paths relate to atomic cuts. Second, the graphical notation might be a helpful metaphor to study the topic. Third, merge paths correspond only to *certain* cuts, much like folding-down or related techniques also correspond only to certain cuts. It is generally accepted that analytic or even atomic cuts should be applied with care in order not to drown in the search space. This is our viewpoint as well. We emphasize one particular property of the transformation t (cf. the figure in the proof of Lemma 5.1): in the cut simulation, the subtree T_C is moved to a different place in the tree. By the bare fact that the considered non-ancestor merge path p is legal in the merge path set containing it, we can be *sure* that the destination of T_C (the C node) contains enough ancestor literals so that T_C *remains* a hyper tableau – that all branches with negative leaves can still be be closed by ancestor merge paths. Clearly, opening branches again would be undesirable as it is unclear if any progress is achieved then. The alternative, forgetting about T_C would cause a lot of recomputation.

Of course, this and other effects and how to avoid them could be formulated as conditions on cuts as well. Non-termination would result if the same atomic cut occurs without bound on a branch.

In sum, these considerations were thought as an argument why things don't get simpler when using cuts, neither conceptually nor algorithmically. In particular, the test for legality would have to have a counterpart when using cuts. A general test for legality is included in [Horton and Spencer, 1997b], that is linear in the size of the clause tree.

6 A Calculus Variant for Circumscription

In the sequel, Γ always denotes some subset of the signature Σ. i.e. a subset of the atoms occuring in a clause set under consideration.

Minimal models are of central importance in various fields, like (logic) programming language semantics, non-monotonic reasoning (e.g. GCWA, WGCWA) and knowledge representation. Of particular interest are Γ-minimal models, i.e. minimal models only wrt. the Γ-subset of Σ. From a circumscriptive point of view, Γ is thus the set of atoms to be minimized, and $\Sigma \setminus \Gamma$ varies.

In this section, we modify the calculus in two ways: first, we are concerned with Γ-minimal models instead of "general" minimal models. That this is a generalisation is easily seen by taking $\Gamma = \Sigma$. Henceforth, by a *minimal* model we mean a Σ-minimal model. Second, we extend the calculus so that *only* (Γ-)minimal models wil be returned as the result of a derivation. This property is referred to as minimal model *soundness*[10] in [Bry and Yahya, 1996].

Definition 6.1 (Γ-Minimal Models)
For any atom set M define the *restriction of M to* Γ as $M|\Gamma = M \cap \Gamma$. In order to relate atom sets M_1 and M_2 define $M_1 <_\Gamma M_2$ iff $M_1|\Gamma \subset M_2|\Gamma$, and $M_1 =_\Gamma M_2$ iff $M_1|\Gamma = M_2|\Gamma$. As usual, the relation $M_1 \leq_\Gamma M_2$ is defined as $M_1 <_\Gamma M_2$ or $M_1 =_\Gamma M_2$. We say that a model $\mathcal{J}$ for a clause set M is *Γ-minimal (for M)* iff there is no model $\mathcal{J}'$ for M such that $\mathcal{J}' <_\Gamma \mathcal{J}$. □

It is easy to see that $\leq_\Gamma$ is a partial order and that $=_\Gamma$ is an eqivalence relation. An obvious consequence of this definition is that every minimal model is also a Γ-Minimal model (but the converse does not hold in general).

From Theorem 5.3 we know that every minimal model can be computed in some branch. Hence, from what was just observed, every Γ-minimal model can be computed as well. However, example 4.1 shows that the calculus defined so far cannot achieve minimal model soundness. It should be noted that all the other mentioned approaches also need an added-on technique to ensure minimal model soundness.

In order to rule out non minimal models some extra test is added. This is done next for the more general case of Γ-minimal models.

6.1 The Calculus

Definition 6.2 (MM-Hyper Tableaux with Merge Paths)
From now on, branches are labeled either as "open", "closed" or with some subset of Γ. In the latter case, b is called a MM-branch, and $\mathrm{MM}(b)$ denotes that set, which is called

[10] Notice that this notion of soundness is only loosely related to the notion of soundness in Section 5.

the *minimal model of b*.

MM-*Hyper tableaux T for* $\mathcal{S}$ *with merge path set* $\mathcal{P}$ – or $(T,\mathcal{P})$ for short – are defined inductively as follows (the text taken literally from Definition 3.1 is set in footnotesize).

Initialization step: as in Definition 3.1.

Hyper extension step with C**:** as in Definition 3.1.

Minimal Model Test If

(i) $(T,\mathcal{P})$ is an open MM-hyper tableau for $\mathcal{S}$ with selected branch b, and

(ii) $[\![b]\!]_{\mathcal{P}}$ is a Γ-minimal model of $\mathcal{S}$

then $(T',\mathcal{P})$ is a MM-hyper tableau for $\mathcal{S}$, where T' is the same as T except that b is labeled with $[\![b]\!]_{\mathcal{P}}|\Gamma$.
If applicability conditions (i) and (ii) hold, we say that the minimal model test inference rule is *applicable* (to b).

Merge path step with p**:** If

(i) $(T,\mathcal{P})$ is an open hyper tableau for $\mathcal{S}$ with selected branch b, and

(ii) $p = mergep(b,b^H)$ is a non-ancestor merge path from b, for some rooted partial branch b^H of T, and

(iii) $last(b^H)$ is not the tail of a merge path in $\mathcal{P}$, and

(iv) $\mathcal{P}\cup\{p\}$ is legal,

then $(T',\mathcal{P}')$ is a hyper tableau for $\mathcal{S}$, where

(i) T' is the same as T, except that b is labeled as closed in T', and every MM-branch b' of T with $[\![b']\!]_{\mathcal{P}\cup\{p\}}|\Gamma \subset \mathrm{MM}(b')$ is labeled as open in T', and

(ii) $\mathcal{P}' = \mathcal{P}\cup\{p\}$.

The definition of derivation and refutation are literally the same, except that we talk about MM-*derivations* and MM-*refutations* now. □

The purpose of the minimal model test rule is to remember that a Γ-minimal model is computed in the selected branch b. Since usually one is interested only in the Γ-subset of models, we keep only the Γ-atoms. Notice that for MM-branches, a hyper extension step is not applicable, because MM-branches are not open. For the same reason merge path steps are also not applicable to MM-branches.

In the course of a derivation a previously computed Γ-minimal model $\mathrm{MM}(b)$ of a branch b might no longer be the same as $[\![b]\!]_{\mathcal{P}}|\Gamma$ because, of a merge path step with head node (for instance) in b. Therefore, the label $\mathrm{MM}(b)$ has to be rejected and the branch has to be opened again for further extension. This is expressed by the new text in (i) in the result description of the merge path step inference rule. Notice, however, that this happens only if some atom $A \in \Gamma$ becomes invisible in $[\![b]\!]_{\mathcal{P}}$, not if some other literal from $\Sigma\setminus\Gamma$ becomes invisible. Thus, some deep merge paths can still be drawn without causing recomputation.

Example 6.3 (MM-hyper tableaux) We continue on Example 3.2-(2). In order to demonstrate the effect of the minimal model test inference rule let now $\Gamma = \{C,E\}$. We start

with tableau 1. For the branch $b_{...}$ the minimal model $[\![b_{...}]\!]_{\mathcal{P}} = \{A, C, \ldots\}$ was supposed. Suppose that E is not contained in that set. Then $[\![b_{...}]\!]_{\mathcal{P}}|\Gamma = \{C\}$ is a Γ-minimal model, because $[\![b_{...}]\!]_{\mathcal{P}}$ is a minimal model. According to the minimal model test inference rule, the branch $b_{...}$ can be labeled with $\{C\}$ then.

Now, consider tableau 2. The merge path p there eliminates the Γ-minimal model candidate in the right branch by closing it. Concerning the left branch $b_{...}$, although A has been removed from its previous interpretation $[\![b_{...}]\!]_{\mathcal{P}} = \{A, C, \ldots\}$, its Γ-minimal model $\{C\}$ has not been changed, i.e. $[\![b_{...}]\!]_{\mathcal{P}}|\Gamma = [\![b_{...}]\!]_{\mathcal{P}\cup\{p\}}|\Gamma = \{C\}$. Consequently the branch label $\{C\}$ has not to be removed and $b_{...}$ has not to be opened again. This is reflected by the result description (i) in the definition of merge path step. If Γ would be taken to, say Σ, the branch $b_{...}$ would have to be opened again and the computation could continue as in Example 3.2-(2) and lead to the tableau 4. □

The Γ-minimal models are thought to be the output of the computation. In order to achieve minimal model completeness and soundness by reading off models from MM-branches only, the old fairness Definition 4.3 has to be changed slightly:

Definition 6.4 (MM-**Fairness**)
Suppose as given some MM-hyper tableau $(T, \mathcal{P})$ for $\mathcal{S}$. A branch b of T is called MM-*finished wrt.* $\mathcal{P}$ iff

(i) b is closed, or

(ii) b is a MM-branch, or else

(iii) the minimal model test inference rule is not applicable to b and every clause $\mathcal{A} \leftarrow \mathcal{B} \in \mathcal{S}$ is redundant in b wrt. $\mathcal{P}$.

The term MM-*unfinished* means "not MM-finished".

MM-*fairness* is defined exactly like fairness, except for the replacement of "derivation" by "MM-derivation" and the replacement of "finished" by "MM-finished". □

According to this definition, the only possibility to be unfair is to terminate an MM-derivation with a selected open branch that could be either labeled with a Γ-minimal model or extended further.

Notice that it is possible, athough not mandatory according to fairness, to apply the minimal model test inference rule to non-finished branches as well. Implementations are thus free to carry out the minimal model test "every now and when". If successful, the branch is turned into a MM-branch and it will not be extended any further.

The next theorem is our main result.

Theorem 6.5 (Minimal Model Soundness and Completeness) *Let f be a model computation selection function and D be a finite, fair* MM*-derivation from satisfiable clause set $\mathcal{S}$ of the* MM*-hyper tableau $(T, \mathcal{P})$. Then*

$$\{\mathrm{MM}(b) \mid b \text{ is a MM-branch of } T\} = \{\mathcal{I}|\Gamma \mid \mathcal{I} \text{ is a } \Gamma\text{-minimal model of } \mathcal{S}\}$$

Furthermore, if $\mathcal{S}$ is unsatisfiable then T is closed (refutational completeness).

Proof. Minimal model soundness – the first theorem statement in the "$\subseteq$"-direction – is an immediate consequence of the applicability condition (ii) in the minimal model test inference rule and the result description (i) in the new merge path step inference rule.

Regarding minimal model completeness – the first theorem statement in the "$\supseteq$"-direction –, suppose to the contrary that for some Γ-minimal model $\mathfrak{I}$ of $\mathcal{S}$ there is no MM-branch of T such that $[\![b]\!]_{\mathcal{P}} =_\Gamma \mathfrak{I}$.

Clearly $\mathfrak{I}|\Gamma \subseteq \mathfrak{I}$ for some minimal model $\mathfrak{I}$ of $\mathcal{S}$. Now, label all MM-branches of T as open and let T' be the resulting tableau. By the soundness lemma (Lemma 5.1) we know that T' contains an open branch b with $[\![b]\!]_{\mathcal{P}} \subseteq \mathfrak{I}$.

In the first case, b is an MM-branch of T. The case $[\![b]\!]_{\mathcal{P}} =_\Gamma \mathfrak{I}$ is impossible by the assumption to the contrary. Hence from $[\![b]\!]_{\mathcal{P}} \neq_\Gamma \mathfrak{I}$ and $[\![b]\!]_{\mathcal{P}} \subseteq \mathfrak{I}$ it follows $[\![b]\!]_{\mathcal{P}} <_\Gamma \mathfrak{I}$. This, however, is impossible by soundness, as it contradicts the given fact that $\mathfrak{I}$ is a Γ-minimal model.

Therefore b is not an MM-branch in T. Since it is open in T' it must be open in T as well. We are given that D is MM-fair. Since f is a model computation selection function, this implies that b is MM-finished.

For this particular b we show next that $[\![b]\!]_{\mathcal{P}} =_\Gamma \mathfrak{I}$ holds. For, suppose to the contrary, that $[\![b]\!]_{\mathcal{P}} \neq_\Gamma \mathfrak{I}$ holds. Again, with $[\![b]\!]_{\mathcal{P}} \subseteq \mathfrak{I}$ it follows $[\![b]\!]_{\mathcal{P}} <_\Gamma \mathfrak{I}$. Since b (of T) is open – i.e. neither closed nor a MM-branch – and MM-finished the minimal model test rule is not applicable and every clause from $\mathcal{S}$ is redundant in b wrt. $\mathcal{P}$. In other words, $[\![b]\!]_{\mathcal{P}} \models \mathcal{S}$. With $[\![b]\!]_{\mathcal{P}} <_\Gamma \mathfrak{I}$ this is a contradiction to the given Γ-minimality of $\mathfrak{I}$. Hence, $[\![b]\!]_{\mathcal{P}} =_\Gamma \mathfrak{I}$.

But then the minimal model test inference rule is applicable to b, because $\mathfrak{I}$ is given as a Γ-minimal model, and so $[\![b]\!]_{\mathcal{P}}$ is a Γ-minimal model as well. Hence b is not finished, contradicting the given fairness of D. So the outermost assumption to the contrary must have been wrong, and the theorem follows.

Refutational completeness is proven as follows: suppose that $\mathcal{S}$ is unsatisfiable but T is not closed. By the minimal model soundness result then T must contain an open branch b (because MM-branches are impossible). Since $[\![b]\!]_{\mathcal{P}}$ is an interpretation and $\mathcal{S}$ is unsatisfiable, $[\![b]\!]_{\mathcal{P}}$ falsifies some input clause from $\mathcal{S}$. But then a hyper extension step is applicable to b with this clause. This contradicts the given fact that D is fair. ■

6.2 Minimal Model Test

Calculi like ours and related calculi need some extra test or device to ensure Γ-minimal model soundness as well. This is hidden in the purely semantical definition of the applicability condition (ii) of the minimal model test inference rule. One option is to use the technique from [Bry and Yahya, 1996] which relies on comparing minimal-model candidates against previously computed minimal models. The same technique was used successfully in [Baumgartner *et al.*, 1997a] for diagnosis applications. To do so within the new calculus, we still have to ensure that the "first" model encountered is a Γ-minimal model.

This technique of turning minimal models into input clauses is not nicely applicable to our case. The reason is that we occasionally have to re-open MM-branches again. Now, when MM-branches would be closed instead, we could no longer distinguish the *reason* why a branch was closed – due to merge path/hyper extension steps or due to the discovery of minimal models. But only in the latter case branches have to be re-opened again. Therefore we decided to introducy the new label type for branches.

The idea of re-using previously computed Γ-minimal models can be used in our case as well. Intentionally, the test for minimal modelship (condition (ii) in the minimal model test rule) is given purely semantically. One strategy is simply to compare the current model candidate against the labels of the previously computed MM-branches. Alternatively, a once computed Γ-minimal model (or simply the Γ-subset thereof) can be remembered elsewhere, say in a table or appropriate data structure. This would have the advantage that the model is preserved during the whole computation.

The drawback of this approach is that it might consume to much memory. In worst case, there are exponentially many minimal models wrt. $|\Sigma|$, which would all have to be kept. Therefore, Ilkka Niemelä developed in [Niemelä, 1996a] to replace the comparison by testing individually each minimal model candidate for minimality by calling a theorem prover instead. The technique was generalized towards circumscriptive reasoning in [Niemelä, 1996b]. A similar technique was proposed in [Aravindan and Baumgartner, 1997] for database applications.

The crucial idea of that test is formulated in the following lemma; it traces back to [Reiter, 1987].

Lemma 6.6 (Minimization Lemma, [Aravindan and Baumgartner, 1997])
Suppose Γ is partitioned as $\Gamma = \Delta \cup \overline{\Delta}$, i.e. $\Delta \cap \overline{\Delta} = \emptyset$. Define $\neg\Delta := \{\neg A \mid A \in \Delta\}$. Let M be a set of formulas. Then

(1a) $M \cup \neg\overline{\Delta} \models \Delta$ and (1b) $M \cup \neg\overline{\Delta}$ is satisfiable

iff

(2a) $M \cup \neg\overline{\Delta} \cup \Delta$ is satisfiable and (2b) Δ is $\subseteq$-minimal for this property.

Property (2b) using an explicit wording shall mean "there is no partition $\Gamma = \Delta' \cup \overline{\Delta'}$ with $\Delta' \subset \Delta$ such that $M \cup \neg\overline{\Delta'} \cup \Delta'$ is satisfiable".

Proof. (1) $\Rightarrow$ (2): Let $\mathfrak{I}$ be a model for $M \cup \neg\overline{\Delta}$, which exists by (1b). By (1a), $\mathfrak{I}$ is a model for Δ as well. Hence (2a) holds. It remains to show (2b). Suppose, to the contrary, that Δ is not a minimal set such that (2a) holds. Hence, there is an $A \in \Delta$ such that $M \cup \neg\overline{\Delta} \cup \{\neg A\} \cup \Delta \setminus \{A\}$ is satisfiable. Trivially, $M \cup \neg\overline{\Delta} \cup \{\neg A\}$ is satsifiable as well. But then, $M \cup \neg\overline{\Delta} \not\models A$, and consequently, $M \cup \neg\overline{\Delta} \not\models \Delta$. Contradiction to (1a). Hence the claim follows.

(1) $\Leftarrow$ (2) Assume that (2a) and (2b) hold. Hence, (1b) follows trivially. Suppose, to the contrary, that (1a) does not hold. That is, $M \cup \neg\overline{\Delta} \not\models A$, for some $A \in \Delta$. Hence, $X := M \cup \neg\overline{\Delta} \cup \{\neg A\}$ is satisfiable. Let $\mathfrak{I}$ be a model for X. According to our convention

of representing models by atom sets, $\mathfrak{I}$ is also implicitly an interpretation for $\Delta \setminus \{A\}$. Next define

$$\Delta' = \{B \in \Delta \setminus \{A\} \mid \mathfrak{I} \models B\}$$
$$\overline{\Delta'} = \overline{\Delta} \cup \{A\} \cup \{B \in \Delta \setminus \{A\} \mid \mathfrak{I} \models \neg B\}$$
$$X' = M \cup \neg\overline{\Delta'} \cup \Delta' \ .$$

Since $\mathfrak{I} \models X$ it follows immediately that $\mathfrak{I} \models X'$ as well. By construction, we also have $\Gamma = \Delta' \cup \overline{\Delta'}$ and $\Delta' \cap \overline{\Delta'} = \emptyset$. From $\overline{\Delta'} \supset \overline{\Delta}$ (by construction) it follows $\Delta' \subset \Delta$, and we have a contradiction to the minimality of Δ (2b). Hence the claim follows. ■

The interesting application of the lemma is this:

Proposition 6.7 (Minimality Test)
Let $(T, \mathcal{P})$ be a MM-hyper tableau for clause set $\mathcal{S}$. Suppose that b is an open branch with $[\![b]\!]_{\mathcal{P}} \models \mathcal{S}$. Then $[\![b]\!]_{\mathcal{P}}$ is a Γ-minimal model of $\mathcal{S}$ if $\mathcal{S} \cup \neg\overline{\Delta} \cup \{\bigvee_{A \in \Delta} \neg A\}$ is unsatisfiable, where $\Delta = [\![b]\!]_{\mathcal{P}}|\Gamma$ and $\overline{\Delta} = \Gamma \setminus \Delta$.

The precondition $[\![b]\!]_{\mathcal{P}} \models \mathcal{S}$ is satisfied in particular for those branches where fairness insists that they are turned into MM-branches, if possible. Hence the minimality test indeed can be used to realize condition (ii) in the minimal model test rule.

Proof. From the definitions it follows immediately that $[\![b]\!]_{\mathcal{P}} \models \neg\overline{\Delta}$ (as defined in the statement of Lemma 6.6). Since we are given that $[\![b]\!]_{\mathcal{P}} \models \mathcal{S}$ it holds $[\![b]\!]_{\mathcal{P}} \models \neg\overline{\Delta} \cup \mathcal{S}$. Hence $\neg\overline{\Delta} \cup \mathcal{S}$ is satisfiable and Condition (1b) in Lemma 6.6) is satisfied. We are given that Condition (1a) holds as well (in a slightly different but equivalent formulation). Hence from now according due to Lemma 6.6 the Properties (2a) and (2b) stated there holds for the sets Δ and $\overline{\Delta}$ determined in the statement of the proposition.

Now we claim slightly more general than in the proposition statement that *any* model $\mathfrak{I}$ of $\mathcal{S}$ with $\mathfrak{I}|\Gamma = \Delta$ is a Γ-minimal model of $\mathcal{S}$. From the definitions it follows immediately that $\mathfrak{I} \models \neg\overline{\Delta}$. The interpretation $[\![b]\!]_{\mathcal{P}}$ is such a model we are talking about, but there can be many more, of course.

Suppose, to the contrary that the claim is false. Hence there is some model $\mathfrak{I}'$ of $\mathcal{S}$ such that $\mathfrak{I}' <_\Gamma \mathfrak{I}$. Since $\mathfrak{I}|\Gamma = \Delta$ this means that $\Delta' = \mathfrak{I}'|\Gamma \subset \Delta$. As before, it follows immediately from the definitions that $\mathfrak{I}' \models \neg\overline{\Delta'}$. Together then conclude that $M \cup \neg\overline{\Delta'} \cup \Delta'$ is satisfiable (as shown by the existence of $\mathfrak{I}'$). This is in plain contradiction to the minimality property of Δ in Lemma 6.6-2. Hence, any model of $\mathcal{S}$ with $\mathfrak{I}|\Gamma = \Delta$ is a Γ-minimal model of $\mathcal{S}$, and so is $[\![b]\!]_{\mathcal{P}}$. ■

The precondition $[\![b]\!]_{\mathcal{P}} \models \mathcal{S}$ is satisfied in particular for those branches where fairness insists that they are turned into MM-branches if possible. Hence the minimality test indeed can be used to realize condition (ii) by reducing the test to a theorem-proving task (unsatisfiability of the stated set). Essentially, this is the approach suggested as the "groundedness test" in [Niemelä, 1996a; Niemelä, 1996b]. In [Aravindan and Baumgartner, 1997] it was employed to ensure minimality of certain database updates. The motivation for this technique is its low (polynomial) memory consumption. The next section contains some more thoughts on this.

6.3 Memory Consumption

In order to guarantee low (polynomial) memory consumption, proof procedures can be based on a "one-branch-at-a-time" approach employing the "local" minimality test from above. Calculi without non-ancestor merge paths would delete closed branches from memory, and once computed models are simply passed to the application environment (without deep merge paths closed branches are of no use).

Now, the same strategy is conceivable in presence of non-ancestor merge paths, namely by allowing to delete MM-branches as well. Of course, deleted branches can no longer be used for merge path steps. From a practical point of view, we propose to fall back to this scheme only when running into memory problems in the course of a derivation with deep merge paths. A half-way compromise would be to allow left/right hooks only, because then only one branch together with its siblings needs to be kept in memory at a time (this suffices because MM-branches are not opened again by adding just hooks and ancestor merge paths).

However, some important applications such as the mentioned diagnosis application typically admit only view minimal models, so that space consumption should not be a problem at all.

7 Conclusions

In this paper we extended previous versions of the hyper tableau calculus by inference rules for merge paths, a device that was originally conceived to speed up refutational theorem proving in the context of clause trees [Horton and Spencer, 1997a]. Our primary goal was to investigate the consequences for model computation purposes. Our main result is therefore a minimal model sound and complete calculus to compute circumscription in the presence of minimized and varying predicates. There is no obstacle to include fixed predicates as well, but we did not do it so far. The motivation was given by the potential to solve a certain problem in diagnosis applications.

We argued that the new calculus generalizes other approaches developed in comparable calculi (folding-up/down, complement splitting), but at the same time we propose to build on known techniques to realize the minimal model test (cf. Section 6.2).

We understand this paper to lay down the theoretical background; how to apply the new technique *practically*, in particular in the envisaged diagnosis domain, is subject to further investigations. Of particular interest will be restricted cases of merge paths that allow for cheap legality tests, as the ALP algorithm of [Horton and Spencer, 1997a], or the Allpaths algorithm of [Sharpe, 1996].

References

[Aravindan and Baumgartner, 1997] Chandrabose Aravindan and Peter Baumgartner. A Rational and Efficient Algorithm for View Deletion in Databases. In Jan Maluszynski, editor, *Logic Programming - Proceedings of the 1997 International Symposium*, Port Jefferson, New York, 1997. The MIT Press.

[Baumgartner *et al.*, 1996] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Artificial Intelligence. European Workshop on Logic in AI, Springer, 1996.

[Baumgartner *et al.*, 1997a] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In *15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 460–465, Nagoya, 1997. International Joint Conference on Artificial Intelligence.

[Baumgartner *et al.*, 1997b] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Tableaux for Diagnosis Applications. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, number 1227 in Lecture Notes in Artificial Intelligence, pages 76–90. Springer, 1997.

[Baumgartner *et al.*, 1999] Peter Baumgartner, Norbert Eisinger, and Ulrich Furbach. A confluent connection calculus. In Steffen Hölldobler, editor, *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*. Kluwer, 1999.

[Baumgartner, 1998] Peter Baumgartner. Hyper Tableaux — The Next Generation. In Harry de Swaart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, number 1397 in Lecture Notes in Artificial Intelligence. Springer, 1998.

[Bry and Yahya, 1996] François Bry and Adnan Yahya. Minimal Model Generation with Positive Unit Hyper-Resolution Tableaux. In Miglioli et al. [1996], pages 143–159.

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

[Horton and Spencer, 1997a] J. D. Horton and Bruce Spencer. Clause trees: a tool for understanding and implementing resolution in automated reasoning. *Artificial Intelligence*, 92:25–89, 1997.

[Horton and Spencer, 1997b] J. D. Horton and Bruce Spencer. Extending the regular restriction of resolution to non-linear subdeductions. In *AAAI-97 — Proceedings of the 14th AAAI National Conference on Artificial Intelligence*, pages 478–483, 1997.

[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.

[Manthey and Bry, 1988] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction, Argonne, Illinois, May 1988*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.

[Miglioli *et al.*, 1996] P. Miglioli, 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.

[Nejdl and Fröhlich, 1996] Wolfgang Nejdl and Peter Fröhlich. Minimal model semantics for diagnosis – techniques and first benchmarks. In *Seventh International Workshop on Principles of Diagnosis*, Val Morin, Canada, October 1996.

[Niemelä, 1996a] Ilkka Niemelä. A Tableau Calculus for Minimal Model Reasoning. In Miglioli et al. [1996].

[Niemelä, 1996b] Ilkka Niemelä. Implementing circumscription using a tableau method. In *Proceedings of the European Conference on Artificial Intelligence*, pages 80–84, Budapest, Hungary, August 1996. John Wiley.

[Reiter, 1987] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, April 1987.

[Sharpe, 1996] D. Sharpe. The allpaths automated reasoning procedure with clause trees. Master's thesis, MCS Thesis, Department of Computer Science, University of New Brunswick, Fredericton, Canada, 1996.