A Disjunctive Positive Refinement of Model Elimination and its Application to Subsumption Deletion[*]

Peter Baumgartner
Universität Koblenz
Institut für Informatik
Rheinau 1
56075 Koblenz
Germany
E-mail: `peter@informatik.uni-koblenz.de`

Stefan Brüning
Technische Hochschule Darmstadt
Fachbereich Informatik
Intellektik
Alexanderstr. 10
64283 Darmstadt
Germany
E-mail: `stebr@intellektik.informatik.th-darmstadt.de`

June 7, 1995

Abstract

The Model Elimination (ME) calculus is a refutational complete, goal-oriented calculus for first-order clause logic. In this paper, we introduce a new variant called *disjunctive positive ME (DPME)*; it improves on Plaisted's positive refinement of ME in that reduction steps are allowed only with positive literals stemming from *disjunctive* clauses.

DPME is motivated by its application to various kinds of *subsumption* deletion: in order to apply subsumption deletion in ME equally successful as in Resolution, it is crucial to employ a version of ME which minimizes ancestor context (i.e. the necessary A-literals to find a refutation). DPME meets this demand. We describe several variants of ME with subsumption, with the most important ones being *ME with backward and forward subsumption* and the T^*-*Context Check*. We relate their pruning power, also taking into consideration the well-known regularity restriction.

All proofs are supplied. The practicability of our approach is demonstrated with practical experiments.

[*]The first author is supported by the DFG within the "Schwerpunktprogramm Deduktion" under grant FU 263/2-2

1 Introduction

The last four decades of research in the field of Automated Deduction have brought forth a couple of exceptionally successful proof systems for first-order logic. In many cases, their success is based on the high inference rates which are achieved using modern computers. Often one can obtain thousands (even millions [van de Riet, 1993]) of true first-order inferences per second. In most cases, however, the problem is not speed but control. While searching for a proof of a theorem, a machine is mostly unable, unlike human beings, to distinguish between relevant and irrelevant information. Since the amount of irrelevant information naturally overwhelms the relevant part, it is impossible, even with the highest inference rates, to examine the whole search space. Therefore, one of the main goals in the field of Automated Deduction is to augment calculi with the possibility to reduce search spaces as much as possible.

Basically, a calculus has to cope with two different kinds of redundancies. First, information can be redundant because of its logical form such that it can be removed without loss of logical information. Second, information can be redundant because it cannot contribute to a successful derivation. Such information can be removed without affecting the satisfiability of the respective formula.

In this paper, we study the problem to avoid logical redundancies during Model Elimination (ME) deductions (e.g. see [Loveland, 1986, Letz *et al.*, 1992, Letz *et al.*, 1994, Baumgartner and Furbach, 1994a] for descriptions of ME calculi). A lot of work done in the field of Automated Deduction to avoid logical redundancies has been carried out in the context of resolution. In resolution based systems this kind of redundancy is traditionally handled by mechanisms which remove tautological and subsumed clauses (e.g. see [Chang and Lee, 1973, Loveland, 1978]). One distinguishes two basic kinds of subsumption: *Forward-subsumption* discards newly generated clauses which are subsumed by already existing clauses whereas *backward-subsumption* removes old clauses which are subsumed by new ones. In [Overbeek and Wos, 1989] it was shown that although subsumption tests can be rather expensive, these mechanisms are very effective to prevent logical redundancy.

Unfortunately, such reduction mechanisms are not applicable in a straightforward manner in model-elimination-like calculi. This is because ME, which – like PROLOG – performs a top-down backward-chaining proof search, does not enumerate derivable clauses but tries to find a proof by enumerating derivations. Needed are therefore mechanisms which prevent *logically redundant derivations*. Within this context, one has to distinguish two basic approaches. The first (and more investigated) one is based on the fact that many subderivations are similar and therefore sometimes interchangeable. Mechanisms which make use of this fact are factorization [Fronhöfer, 1985], lemma handling [Fronhöfer and Caferra, 1988], and caching [Astrachan and Stickel, 1992].

In what follows we shall pursue the second basic approach which aims at avoiding a derivation if the possibility to extend it to a refutation would imply the same possibility for a different and (hopefully) smaller derivation.

There already exist some techniques following this general approach. Popular ones are the so-called *Identical Ancestor Check* and its generalization, *regularity*, which are successfully used in several theorem provers (e.g. see [Stickel, 1988, Letz *et al.*, 1992]). These refinements prune a derivation if some literal is identical to one of its ancestors. Other restrictions on ME to prevent redundant derivations are *cut-freeness* and *subsumption-freeness* [Letz, 1993] (which can be seen as restricted versions of tautology and subsumption deletion in resolution based systems). The first one ensures that tableaux do not contain tautological tableau clauses. The latter one demands that no tableau clause is properly subsumed by some clause of the input set.

A further possibility to avoid redundant derivations, which is mostly neglected in connection with top-down backward-chaining calculi, is the use of subsumption. As shown in [Besnard, 1989, Bol *et al.*, 1991], SLD-resolution, which can be seen as a restricted variant of ME, can be combined with a technique which compares (via subsumption) sets of open goals.[1] Whenever the set of open goals after applying a sequence of inference steps $d_1, \ldots, d_n$ is subsumed by some former set of open goals (that is the set of open goals after some d_j with $j < n$), the last inference step, d_n, can be withdrawn.

Unfortunately, a straightforward application of this kind of forward-subsumption to ME only preserves completeness in case Horn clause sets are considered. Roughly speaking, this is due to the fact that ancestor goals which may become important to perform reduction steps, are ignored. Hence, a first step to retain completeness would be to compare entire ME tableaux. However, it it becomes clear quite immediately that without further refinements such a proceeding would be (rather) useless: A ME tableau T_n generated after a a sequence $D = d_1, \ldots, d_n$ of inference steps usually contains more ancestor goals than the tableau T_j generated by $d_1, \ldots, d_j$ where $j < n$. Since every ancestor of an open goal is a potential candidate for the application of a reduction step, one cannot prune D unless, roughly speaking, each of the "additional" ancestors in T_n also occur in T_j in the corresponding tableau branches. This, however, only happens very rarely to be the case.

In this paper, we propose two approaches to overcome this problem. Both of them provide criteria which allow to identify ancestor goals (i.e. positive A-literals in Loveland's terminology) that can be safely ignored as potential candidates for the application of reduction steps. Hence, in case these criteria tell us that, for instance, none of the ancestors in T_n is necessary for the application of reduction steps, we can safely prune D if the set of open goals of T_j subsumes the set of open goals of T_n.

The first and more important approach is based on the *positive refinement* for ME. This refinement restricts reduction steps to those that use a positive ancestor goal [Plaisted, 1990] and therefore allows to ignore all negative ancestors. However, one can do better. In Section 3, we prove that it is additionally possible (without loosing completeness) to ignore positive ancestors which originate from Horn clauses. Thus, the sole ancestors which have to be taken into account by a subsumption-based pruning technique are non-negated literals which stem from non-Horn clauses (called *disjunctive clauses*).

The second approach is to predetermine which ancestors of a goal G *cannot* be used for reduction steps in order to solve G. This is achieved by the concept of *reachability* originally proposed in [Neugebauer, 1992] (in a different context). Roughly speaking

[1]Note, that in terms of resolution, the set of open goals is in fact a resolvent.

the idea is as follows: in a preprocessing step it is determined which literals are reachable from G, i.e. which literals (modulo additional substitutions) may occur as subgoals of G. Then, one can safely ignore any ancestor A of G in case no literal, which can be made complementary to A (via unification), is reachable from G.

With these two approaches at hand, we define several pruning techniques based on subsumption for ME. Besides a generalized version of the aforementioned variant of forward-subsumption we also introduce backward-subsumption, which, similarly to the concept of backward-subsumption in resolution based calculi, prunes a derivation D_1 if it is subsumed (in the above sense) by a derivation D_2 which is generated after D_1 in the course of the deduction process (Section 4).

Furthermore, we propose a variant of forward subsumption which only compares one open goal with one of its ancestor goals, rather than comparing sets of literals ("T-Context Check", Section 5). Roughly speaking, the idea is to prune a derivation if some open goal is an instance of one of its ancestor goals. Clearly, such a proceeding only preserves completeness if variable dependencies with other goals are taken into account. The resulting pruning technique is (in case Horn clause sets are considered) a true generalization of the loop check proposed in [Besnard, 1989] and of the aforementioned identical ancestor check. Further, we shall illustrate in Section 4 that it does not suffer from one main problem of other subsumption-based techniques, namely a dependence of the employed search strategy.

Finally, it is worth emphasizing that the techniques proposed in this paper not only allow to reduce the search space in order to find proofs more quickly. Additionally they make it possible to detect that formulas *cannot be proven.* This is particularly interesting for applications where false conjectures or incorrect theorems have to be identified. Note, that many other techniques, for instance factorization or the use of lemmata, do not contribute to this problem.

The paper is structured as follows: In Section 2 we provide some basic definitions to make this paper self-contained. Section 3 is devoted to the introduction of the positive refinement and its generalization, the so-called *disjunctive positive refinement.* There, the completeness of ME in combination with the disjunctive positive refinement is proved. In Section 4 and Section 5 we introduce the aforementioned subsumption-based pruning techniques. We discuss their relation as well as their connection to other known refinements of ME. In Section 6 we illustrate the usefulness of our techniques by presenting some experimental data. In Section 7 and Section 8 we discuss our results and point out some future research perspectives. Figure 10 in Section 7 is a graphical summary of our results; it might be a good idea to occasionally have a look at it for orientation during reading.

2 Preliminaries

In what follows, we assume the reader to be familiar with the basic concepts of first-order logic. A *clause* is a multiset of literals. As usual, the variables occurring in clauses are considered implicitly as being universally quantified, a clause is considered logically as a disjunction of literals, and a clause set is taken as a conjunction of clauses. A clause is *Horn* iff it contains at most one positive literal. Let L^d denote the complement of a literal L. Two literals L and K are *complementary* if $L^d = K$. The set of variables occurring in an expression E is denoted by $var(E)$.

Throughout this paper, we consider a variant of ME which uses so-called ME tableaux as basic proof objects (e.g. see [Letz *et al.*, 1992, Letz *et al.*, 1994]), rather than ME chains [Loveland, 1986]. An alternate, and equally usable approach would have been to follow [Baumgartner and Furbach, 1993] and view ME as a transformation on path sets.

Definition 2.1 (Literal tree) A *literal tree* is a pair (t, λ) consisting of an ordered tree t and a labeling function λ assigning literals or multisets of literals to the non-root nodes of t. The *successor sequence* of a node N in an ordered tree t is the sequence of nodes with immediate predecessor N, in the order given by t. **(End Definition)**

Definition 2.2 (Tableau) A *(clausal) tableau* T of a set of clauses $\mathcal{S}$ is a literal tree (t, λ) in which, for every successor sequence $N_1, \ldots, N_n$ in t labeled with literals $K_1, \ldots, K_n$, respectively, there is a substitution σ and a clause $\{L_1, \ldots, L_n\} \in \mathcal{S}$ with $K_i = L_i\sigma$ for every $1 \leq i \leq n$. $\{K_1, \ldots, K_n\}$ is called a *tableau clause* and the elements of a tableau clause are called *tableau literals.*

A tableau is called *model elimination tableau (ME tableau)* if each inner node N labeled with a literal L has a leaf node N' among its immediate successor nodes which is labeled with the literal L^d. **(End Definition)**

Let $T\delta$ denote the tableau which is obtained from a tableau T by application of the substitution δ to all literals of T. Furthermore, given a tableau T containing some node N, we often denote the literal attached to N in T by $L(N, T)$.

Definition 2.3 (Branch, Open and Closed Tableau, Open Goal, Frontier) A *branch* of a tableau T is a sequence $N_1, \ldots, N_n$ of nodes in T such that N_1 is the root of T, N_i is the immediate predecessor of N_{i+1} for $1 \leq i < n$, and N_n is a leaf of T. A *branch is complementary* if the labels of $N_1, \ldots, N_n$ contain some literal L and its complement L^d. In order to distinguish the simple presence of a complementary branch and the detection of this fact, we allow to label branches as either *open* or *closed*. A tableau is *closed* if each of its branches is closed, otherwise it is *open*. Let $OB(T)$ denote the set of open branches of T.

Let b be an open branch of a ME tableau T. If L is the label of the leaf node of b, then L is called an *open goal* of T. Further, the set of leaf nodes in T which are labeled with open goals is called the *frontier* of T. **(End Definition)**

The motivation to distinguish between complementary and closed tableau branches will be given after the ME calculus has been defined. Given a branch $N_1, \ldots, N_n$ we sometimes say that N_i *dominates* N_j if $i < j$.

Theorem 2.4 ([Letz, 1993]) *Let $\mathcal{S}$ be clause set. $\mathcal{S}$ is unsatisfiable iff there exists a closed ME tableau of $\mathcal{S}$.*

Based on the above definitions we now introduce the inference steps of ME. Their purpose is to allow for systematical construction of ME tableaux. Throughout the following definitions, let $\mathcal{S}$ be a set of input clauses.

Definition 2.5 (Initialization step) Let N be the root of a one-node tree. Select a new variant $\{L_1, \ldots, L_n\}$ of a clause $C \in \mathcal{S}$, attach n new successor nodes to N, and label them with $L_1, \ldots, L_n$, respectively. C is called *top-clause*. **(End Definition)**

Definition 2.6 (Extension step) Select a leaf node N of an open branch labeled with literal L. Let $C' = \{L_1, \ldots, L_n\}$ be a new variant of a clause $C \in \mathcal{S}$ such that there exists a MGU σ and a literal $L_i \in C'$ with $L^d\sigma = L_i\sigma$. Then attach n new successor nodes $N_1, \ldots, N_n$ to N, label them with $L_1, \ldots, L_n$, respectively, and apply σ to all tableau literals. Finally, the new branch with leaf node N_i is marked as closed.

We call N_i an *extension node*, and each element of $\{N_1, \ldots, N_n\} - \{N_i\}$ a *non-extension node*. A literal attached to an extension node is called *extension literal*, otherwise *non-extension literal*. **(End Definition)**

Definition 2.7 (Reduction step) Select a leaf node N' of an open branch b labeled with literal L'. If there is a dominating node N on b labeled with literal L such that there exists a MGU σ with $L'\sigma = L^d\sigma$, then apply σ to all tableau literals and mark b as closed. **(End Definition)**

Note that as an immediate consequence of these definitions we have that a branch is closed only if it is complementary (but not necessarily vice versa).

Definition 2.8 (Model Elimination) A sequence of $T_1, \ldots, T_n$ of ME tableaux is called a *ME derivation* for a clause set $\mathcal{S}$ (called the *set of input clauses*) if T_1 is obtained by an initialization step, and for $1 < i \leq n$, T_i is obtained either by an initialization step, or by one single application of a reduction step or an extension step to some node in T_{i-1}. Furthermore, each clause used in an extension step has to be a variant of an element of $\mathcal{S}$. A ME derivation is called a *ME refutation* if it generates a closed tableau.

It turns out to be practical to confuse a derivation, i.e. a sequence of tableaux, with the process generating it. We will thus write $d_1, \ldots, d_n$ instead of $T_1, \ldots, T_n$, where the d_is shall denote the used instances of the respective inference rules to obtain the T_is.

Let L be an open goal attached to a node N in a ME tableau T. A *ME sub-derivation* D for N (or L) is a sequence of derivation steps where the first element of D selects N and each further element selects a descendant of N. D is called a *ME subrefutation* if after applying D to T, each branch containing N is closed.

We simply say "*(sub)derivation*" instead of "ME (sub)derivation" and "*(sub)refutation*" instead of "ME (sub)refutation".

A derivation D is called *unrestricted* if the substitution used to apply the elements of D need not to be MGUs. **(End Definition)**

Now we can motivate the explicit closing of a branch: firstly, it is natural in view of a proof procedure to make the detection of a complementary branch explicit; secondly, it facilitates the definition of the disjunctive positive variant of ME below; thirdly, (a technical motivation) the lifting of derivations from the ground level to the general level is less complicated, because otherwise reduction steps at the general level become necessary which do not have a counterpart at the ground level; fourthly, if the distinction were not made, it would be *mandatory* to check the selected branch for complementarity because a complementary (and hence closed) branch cannot be selected for a derivation step.

Figure 1 contains a sample subrefutation for the node $\neg A$ in tableau $\boxed{1}$.

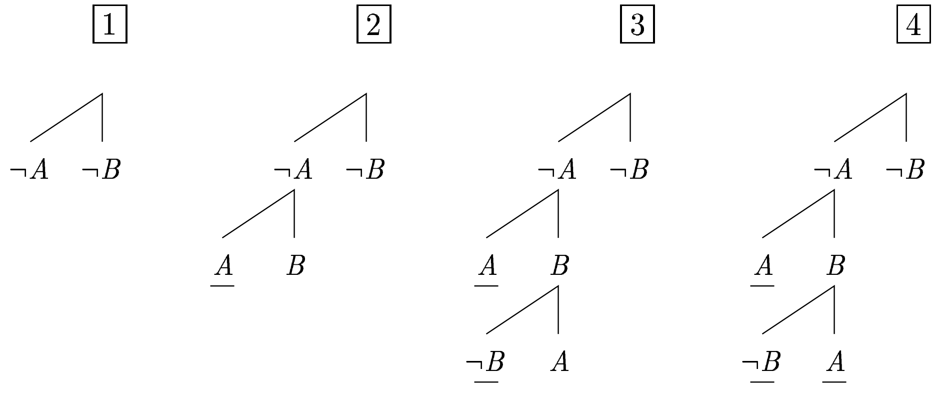


Figure 1: A subrefutation for $\neg A$. It is the shortest possible one for the input clause set $\{\neg A \vee \neg B,\ \neg A \vee B,\ A \vee \neg B,\ A \vee B\}$. Tableaux $\boxed{2}$ and $\boxed{3}$ are obtained by extension steps, and tableaux $\boxed{4}$ by a reduction step. All substitutions involved are the empty substitution. The leaf literals of closed branches are underlined. Note that although the branch with leaf A in tableau $\boxed{3}$ is complementary, it is not closed.

Theorem 2.9 ([Letz, 1993]) *Given any ME tableau T of a clause set $\mathcal{S}$ there is a ME derivation of $\mathcal{S}$ generating a tableau T' such that T is an instance of T'.*[2]

Definition 2.10 ((Local) Computation Rule) A *computation rule* r is a mapping assigning to each open tableau T a leaf node N in T which is labeled with an open goal. N is called the node selected by r. Alternatively, we often say that the literal attached to N is selected by r.

A *derivation via a computation rule* r is a derivation $d_1, \ldots, d_n$ such that for all $1 \leq i < n$, d_{i+1} is applied to the node selected by r in the tableau generated by $d_1, \ldots, d_i$.

We call a computation rule r *local* if every derivation $d_1, d_2, \ldots$ generated via r satisfies the following property: whenever a node N is selected for the application of a derivation step d_i, then a brother node N' of N is selected for a derivation step d_j with $j > i$ only in case each path containing N has been closed previously. Further, we require local computation rules to be stable under substitution, that is a node is selected in a tableau T by r if and only if it is also selected in $T\sigma$ for every substitution σ. **(End Definition)**

For instance, a computation rule which always selects the "leftmost" open branch is local. Likewise, always selecting a longest open branch is also local.

[2]A tableau T is an *instance* of another tableau T' iff there is a substitution σ such $T'\sigma = T$.

Proposition 2.11 ([Letz, 1993]) *Any closed ME tableau for a set of ground clauses can be constructed with any possible computation rule.*

3 A Disjunctive Positive Refinement of Model Elimination

In this section we will introduce a *disjunctive positive refinement* of ME. This is a strengthening of the *positive refinement* of [Plaisted, 1990]. Let us first motivate it by its application for subsumption deletion.

When attempting to define a subsumption relation among ME tableaux, one soon recognizes that the situation is more complicated than in resolution. In order to explain this let us take the view of a ME tableau as a generalized clause, whose literals are the tableau's leaf literals (thus, we are talking about the "frontier" in the sense of Def. 2.3). The ancestor literals then can be thought of as additional literal lists attached to the clause literals (See [Baumgartner and Furbach, 1993] for a more detailed treatment). Clauses as used in Resolution then can be seen as frontiers of ME tableaux.

In order to adapt the notion of subsumption of resolution[3] to ME, a first idea is to apply subsumption among tableaux frontiers to discard subsumed tableaux. This approach is appealing because no ancestor context needs to be examined. Unfortunately, completeness is lost then, because some ancestor context present only in the discarded tableaux might be necessary to complete the proof. Thus, ancestor context has to be obeyed for the subsumption check. On the other hand, if the *whole* ancestor context for every literal is obeyed, then chances are low to apply subsumption successfully very often.

In order to overcome these problems we propose to use a variant of ME which minimizes ancestor literals. In [Plaisted, 1990] D. Plaisted has shown that it suffices for completeness to keep only *positive* literals as ancestors. Using this calculus a larger subsumption relation can be defined, since only the positive ancestor literals have to be obeyed for the subsumption test. We will further improve on this by restricting ancestor context to those positive literals which stem from disjunctive clauses[4]. For example, if a leaf literal A is extended with the clause $B \vee \neg A$ then both ordinary ME and Plaisted's positive refinement will keep B in the ancestor context (and thus allowing reduction steps to B) of the subrefutation of B, while in our *positive disjunctive refinement* it will not be kept. These cases occur every time when a definite clause is used with a "body" literal as entry point. In order to state a positive example consider $C \vee B \vee \neg A$; then after an extension step to A both C and B have to be kept for reduction steps in the subrefutations of C and B.

Since only these positive disjunctive literals are needed to get a complete calculus, it is thus sufficient to restrict subsumption to obey these literals only. A detailed treatment on this can be found in Section 4.

The possibility to restrict ancestor steps to disjunctive positive literals has been observed independently from our work in [Mayr, 1995]. This result is obtained as

[3]Recall that a clause C *subsumes* a clause D iff for some substitution σ we have $C\sigma \subseteq D$, where a clause is a *multi*set of literals.

[4]By a *disjunctive clause* we mean a clause which contains at least two positive literals

a by-product of the completeness proof there. That proof proceeds by transforming a given restart ME [Baumgartner and Furbach, 1994a] refutation into an ordinary ME refutation in such a way that the absence of reduction steps to non-disjunctive positive literals carries over to the transformed proof (below we will give another, more direct proof).

In the following we will formally introduce the positive disjunctive refinement and prove its completeness.

3.1 Definition of the Disjunctive Positive Refinement

As indicated above, the information whether a certain literal stems from a positive disjunctive clause is crucial. The next definition accounts for this formally.

Definition 3.1 A *disjunctive* clause is a clause which contains at least two positive literals. The complementary class of Horn clauses is also called *non-disjunctive* clauses in this context. We think of the positive literals attached to tableau nodes as being labeled respectively: a tableau node N_i being part of a successor sequence $N_1, \ldots, N_n$ is labeled as N_i^+ iff the attached tableau clause $\{K_1, \ldots, K_n\}$ is disjunctive and K_i is a positive literal. The (type of a) node N^+ is also classified as *disjunctive positive*.

As a notational convention, we will allow labeling the literal instead of the node. Further we often call literals disjunctive positive, if they are attached to disjunctive positive nodes. **(End Definition)**

Based on this we can define the disjunctive positive refinement:

Definition 3.2 (Disjunctive Positive Reduction Step and Disjunctive Positive Refinement) In the following let N be an open goal node in a tableau T. Suppose a computation rule r as given.

We are going to define a restricted version of "reduction step" (cf. Definition 2.7): if N' is selected in T by r, and if there is a dominating positive disjunctive node N^+ of N' on the same branch, and if there exists a MGU σ with $L(N', T)\sigma = L(N^+, T)^d\sigma$, then apply σ to all tableau literals. Such reduction steps are called *disjunctive positive reduction steps (via r)*.

The definition of extension step (Definition 2.6) via r remains unchanged. In particular, the type of the extension node is understood to be disregarded, i.e. both disjunctive positive and non-disjunctive positive nodes have to be extended. As in the *disjunctive positive reduction step*, only extension of a *selected* node is allowed.

Finally, the notion of derivation (Definition 2.8) is adapted to incorporate these changes, and we will speak of *disjunctive positive* derivations and refutations. The calculus is termed *disjunctive positive refinement* of ME (DPME). **(End Definition)**

Figure 2 contains a sample refutation.

Note 3.3 (Extra literals) Since the non-disjunctive positive literals and the negative literals of a branch cannot be subject to reduction steps, it would in a strict sense not be necessary to store them along a refutation. Nevertheless it may be advisable to do so. One reason is that occasionally allowing (ordinary) reduction steps helps to find a refutation more quickly. For instance, in Figure 2, tableau $\boxed{3}$ the path ending in

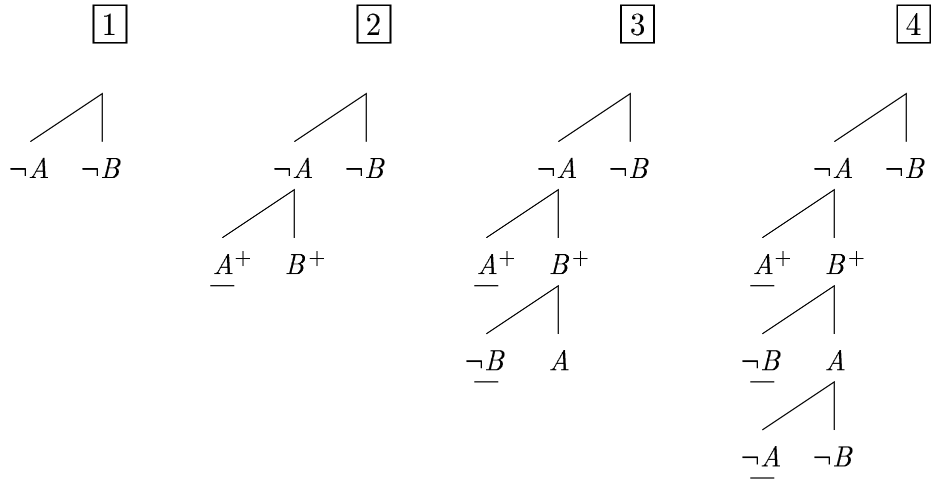


Figure 2: A subrefutation for $\neg A$. It is the shortest possible one for the input clause set $\{\neg A \vee \neg B,\ \neg A \vee B,\ A \vee \neg B,\ A \vee B\}$. Tableau $\boxed{2}$, $\boxed{3}$ and $\boxed{4}$ are obtained by extension steps. Note that the node A in tableau $\boxed{3}$ cannot be closed by a disjunctive positive reduction step, but the node $\neg B$ in $\boxed{4}$ can. A similar subrefutation exists for $\neg B$.

A could immediately be closed by an (ordinary) reduction step. Indeed, a respective strategy for the *restart* version of ME (see Note 3.5 below) turned out to be essential in practice [Baumgartner and Furbach, 1994b].

Another application concerns the possibility of regularity checks based upon the extra literals. Although the disjunctive positive refinement is not compatible with full regularity[5] ([Letz, 1993] has shown that regularity is incompatible even to Plaisted's positive refinement), it is possible to safely combine it with *blockwise regularity*. **(End Note)**

Definition 3.4 (Blockwise Regularity, Disjunctive Regularity) Let b denote a branch $N_0, N_1, \ldots, N_n$ of a ME tableau. A *block* of b is a maximal sequence $N_i, \ldots, N_j$ with $1 \leq i < j \leq n$ such that none of its elements is disjunctive positive.

A branch in a ME tableau is called *blockwise regular* if it contains no block with two different nodes that are labeled with the same literal. A ME tableau is called *blockwise regular* if all its open branches are blockwise regular.

Further, a branch is is called *disjunctive regular* if it does not contain two different positive disjunctive nodes that are labeled with the same literals. A ME tableau is called *disjunctive regular* if all its open branches are disjunctive regular.

A ME deduction is called *blockwise (and disjunctive) regular* if each derivation generating tableaux which violate blockwise (or disjunctive) regularity is pruned. **(End Definition)**

[5]The regularity restriction forbids to generate tableaux containing a branch with two identical literals on it.

Note 3.5 (Restart Model Elimination) The completeness proof below is similar to that of *restart ME* [Baumgartner and Furbach, 1994a]. This suggests to impose restrictions defined for restart ME also on disjunctive positive ME. Indeed, blockwise regularity holds for both calculi. However, while the analogue to disjunctive regularity — a "global" regularity wrt. all positive literals in a branch — *is* compatible to restart ME, things are more involved for disjunctive regularity. Since we did neither find a respective proof nor a counterexample this problem must remain open.

Another feature of restart ME is the possibility to introduce a *selection function*. A selection function is a function which maps a clause to one of its positive literals. Now, in restart ME, extension steps can be restricted in such a way that *only* the selected literal may be used as extension literal. It soon becomes obvious that for disjunctive positive model elimination at least the negative literals of a clause must be selected as well. But, unfortunately, even such relaxed selection functions cause incompleteness. The canonical counterexample consists of the clauses $\{\neg A,\ A \vee B,\ \neg B \vee \neg C,\ A \vee C\}$ and a selection function which selects A in both positive clauses. Then no refutation exists. **(End Note)**

3.2 Soundness and Completeness

While soundness of the disjunctive positive refinement follows immediately from the soundness of ordinary ME, completeness is harder to establish.

Theorem 3.6 (Completeness) *Let $\mathcal{S}$ be an unsatisfiable clause set and r be a computation rule. Then there exists a blockwise regular disjunctive positive ME refutation via r of S with some negative clause from $\mathcal{S}$ as top clause.*

The proof of the theorem proceeds by assuming a set $\mathcal{S}^g$ of ground instances of clauses of S which is unsatisfiable. Such a set exists according to the Skolem-Herbrand-Löwenheim theorem (see e.g. [Gallier, 1987] for a proof). Without loss of generality we can think of $\mathcal{S}^g$ as being minimal unsatisfiable. Furthermore, $\mathcal{S}^g$ must contain a negative clause G (because otherwise $\mathcal{S}^g$ would be satisfiable). Then by ground completeness (which is to be proven below) a refutation on the ground level with top clause G exists. However, this proof does not give us the claimed independence of the computation rule. A respective result (for the ground case) was proven explicitly in [Baumgartner, 1994]. This proof works for *any* set of inference rules which work "locally", which means that only one single branch is affected by the inference rules. Since this applies in our case we can take the respective result for our calculus as granted.

The lifting to the first-order case can be carried out by using standard techniques. See [Baumgartner *et al.*, 1995] for a lifting proof for ME. In particular, independence of the computation rule is an easy consequence from the result for the ground level (because if a refutation at the first-order level via some computation rule r would not be possible, it would neither be possible at the ground level, where corresponding nodes are selected).

Ground completeness reads as follows:

Lemma 3.7 (Ground Completeness) *Let $\mathcal{S}$ be a minimal unsatisfiable ground clause set, and $G \in \mathcal{S}$. Then there exists a blockwise regular disjunctive positive ME refutation of S with top clause G.*

Informally, the proof is by splitting the non-Horn clause set into clause sets which are "more Horn" and assuming by the induction hypotheses ME refutations of these sets, and then assembling these refutations into the desired disjunctive positive ME refutation. The base case in the induction proof deals with Horn clause sets.

Reduction steps come in when assembling back the splitted refutations. More precisely, extension steps with split unit clauses have to be replaced by reduction steps to the literals where the split occurred. While blockwise regularity carries over from the regularity of the base case (Horn case), the disjunctive regularity is more complicated. Since we did neither find a respective proof nor a counterexample this problem must remain open.

As mentioned, the base case consists of Horn sets. The respective result reads as follows:

Lemma 3.8 (Ground Completeness for Horn clause sets without reduction steps) *Let $\mathcal{S}$ be a minimal unsatisfiable ground Horn clause set and $G \in \mathcal{S}$ be a (not necessarily negative) clause. Then there exists a blockwise regular ME refutation of S with top clause G where no reduction step is applied.*

It is well-known that ME is complete for Horn clause sets without reduction steps for *negative* top clauses. In fact, the result follows immediately from the completeness of ME for general clause sets and the simple observation that reduction steps are not possible at all in that special case. However, the situation is different if *any* (not necessarily negative) clause is allowed as a top clause, because the simply syntactic argument is no longer valid. Thus, Lemma 3.8 is non-trivial.

A completeness result similar to Lemma 3.8 has been proven in [Antoniou and Langetepe, 1994] in the context of SLD-Resolution. However, they do not consider regularity. Thus we need a new proof.

Unfortunately, we cannot demand a stronger version of regularity which also considers already closed branches. This difference is crucial for Lemma 3.8. Consider for instance the Horn clause set

$$\{A,\ \neg A \vee B,\ \neg B \vee \neg A\},$$

and select A as top clause. Both existing refutations *without* reduction steps have to violate the full regularity restriction. Nevertheless, regularity wrt. open branches holds.

Proof. (Lemma 3.8) If G is a negative clause we will rely on earlier completeness proofs (e.g. [Baumgartner, 1994]). Thus suppose from now on that G is a definite clause. By the completeness theorem from [Baumgartner, 1994], we know that there exists a ME refutation R of $\mathcal{S}$ with top clause G. Now we proceed in two stages: we will show (1) how possibly applied reduction steps in R can be transformed away, yielding R', and then, (2) how regularity violations in R' can be eliminated.

Ad (1): Let T_R be the closed tableau generated by R, and let B be the multiset of closed branches of T_R. The top clause can be written as $G = A \vee \neg G_1 \vee \cdots \vee \neg G_n$. Let

R_A be the subrefutation of A. For syntactical reasons, reduction steps can only occur in R_A. Let $B_A \subseteq B$ be the multiset of branches generated by R_A. Also for syntactical reasons, every branch in $b \in B_A$ is of the form $b = K_1, \ldots, K_i, K_{i+1}, \ldots, K_m$ where $m > 1, i \geq 1$, $K_1, \ldots, K_i$ $(K_1 \equiv A)$ are positive literals and K_{i+1} is a negative literal. Furthermore, b is closed by a reduction step if and only if K_m is a negative literal, and in this case ${K_m}^d = K_j$ for some $j \in \{1, \ldots, i\}$. In words, reduction steps can occur only from negative to positive literals, the latter of which are all in the prefix of the branch being closed.

Next, we locate the reduction step to a "bottommost" positive literal, because this is the one to be transformed away. In this transformation new reduction steps will possibly be introduced, but each of them reduces to a dominating node. This gives us the termination of the procedure.

For the formal treatment we define for a branch $b \in B_A$ in the form above the functions $f_b : \{K_1, \ldots, K_i\} \mapsto \{0, 1\}$[6] and, based on this, the function f:

$$f_b(K) = \begin{cases} 1 & \text{if } b \text{ is closed by a reduction step to } K \\ 0 & \text{else} \end{cases}$$
$$f(b) = f_b(K_i), \ldots, f_b(K_1)$$

Thus, $f(b)$ maps b to a sequence of 0s and 1s, containing at most one occurrence of 1 (depending whether b is closed by a reduction step or not). The "reverse" ordering of indices is motivated by the way branches are to be compared: for two branches $b_1, b_2 \in B_A$ we define $b_1 \prec b_2$ iff b_1 is strictly smaller in the lexicographical extension of the usual ordering "$<$" on naturals. Informally, branches get smaller as reduction steps are applied to more "topmost" literals. The ordering "$\prec$" is extended to branch multisets by defining $B' < B$ iff $F(B'_A) \prec\!\!\prec F(B_A)$ where $F(B_A) = \{\!\{f(b) \mid b \in B_A\}\!\}$ and "$\prec\!\!\prec$" denotes the multiset extension of "$\prec$". When we write $R' < R$ for (sub-)refutations we mean the corresponding generated multisets of branches.

It is well-known that with "$<$" being well-founded, so is "$\prec$" and "$\prec\!\!\prec$". Thus we can apply well-founded induction.

Base case: every element in $F(B_A)$ is a sequence $0, \ldots, 0$. In other words, the subrefutation R_A is constructed without any reduction step. Since, if at all, reduction steps can occur only in R_A, R does not contain reduction steps either. Thus the result holds.

Induction step: As the induction hypothesis, suppose the result to hold for all refutations R' with $R' < R$.

$F(B_A)$ contains an element $f(b) = f_b(K_i), \ldots, f_b(K_1)$ for some branch $b \in B_A$ which is not a sequence of 0s, i.e. $f_b(K_j) = 1$ for some $j \in \{1, \ldots, i\}$. By construction of tableaux, K_j is contained in an input clause

$$C = K_j \vee \vec{c}$$

where $\vec{c}$ denotes the rest literals of C.

We distinguish two cases:

[6]To be precise, in the domain of f_b we mean the occurrences, i.e. the nodes, but not the labels.

Case $j = 1$ ($f(b) = 0, \dots, 0, 1$)[7], i.e. a reduction step to the topmost positive literal of b, K_1 ($\equiv A$), occurred (cf. Figure 3). In this case we first replace in R_A this reduction step from ${K_i}^d$ to K_1 by an extension step with $K_1 \vee \vec{c}$. Let us by $R_{\vec{c}}$ denote the collection of subderivations of the (negative!) literals of $\vec{c}$. These exist, because we are given that R is a refutation, and furthermore, because $K_1 \vee \vec{c}$ is the top-clause, none of these uses a reduction step. Then we append the subrefutations $R_{\vec{c}}$ in order to close the newly introduced open leafs $\vec{c}$. Call the resulting subrefutation R'_A. It holds $R'_A < R_A$ (formally: the element $0, \dots, 0, 1 \in F(B_A)$ has been replaced by finitely many sequences of 0s, which is decreasing). This in turn is a consequence of the facts that the reduction step has been replaced by an extension step, and that $R_{\vec{c}}$ does not need reduction steps.

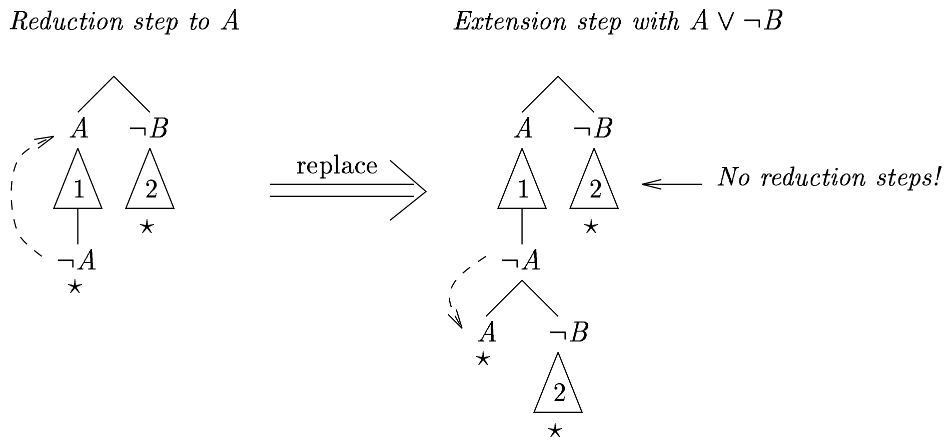


Figure 3: A prototypical example for case 1 in the proof of lemma 3.8: a reduction step to the topmost positive literal occurred.

Case $j > 1$ ($f(b) = 0, \dots, 0, 1, 0, 0, \dots, 0$), i.e. a reduction step occurred to a positive literal in b, which is not topmost (cf. Figure 4). We have to rewrite C as

$$C = K_j \vee \neg L \vee \vec{c'}$$

where $\neg L$ is the (negative!) literal in C which was used as extension literal when C was entered into the tableaux by extending K_{j-1}.

In this case we first replace in R_A the reduction step from ${K_i}^d$ to K_j by an extension step with $K_j \vee \neg L \vee \vec{c'}$. Then we close the resulting branch $b, \neg L$ by a reduction step to K_{j-1}. We still have to close the rest literals $\vec{c'}$. For this let us by $R_{\vec{c'}}$ denote the collection of subderivations of the (negative!) literals of $\vec{c'}$. These exist, because we are given that R is a refutation, and furthermore, any reduction steps used therein must reduce to a predecessor node of K_j. Next we append the subrefutations $R_{\vec{c'}}$ in order to close the newly introduced open leafs $\vec{c}$. Call the resulting subrefutation R'_A. It holds $R'_A < R_A$ (formally: the element $f(b) = 0, \dots, 0, 1, 0, 0, \dots, 0 \in F(B_A)$ has

[7]The expression $0, \dots, 0$ stands for a sequence of zero or more 0s.

been replaced by (a) a sequence of 0s, stemming from the extension step, and (b) a sequence $0, \ldots, 0, 0, 1, 0, \ldots, 0 \prec 0, \ldots, 0, 1, 0, 0, \ldots, 0$ stemming from the reduction step from $\neg L$, and (c) by some sequences stemming from the subrefutations $R_{\vec{c'}}$, each of them being strictly smaller wrt. $\prec$ than $f(b)$).

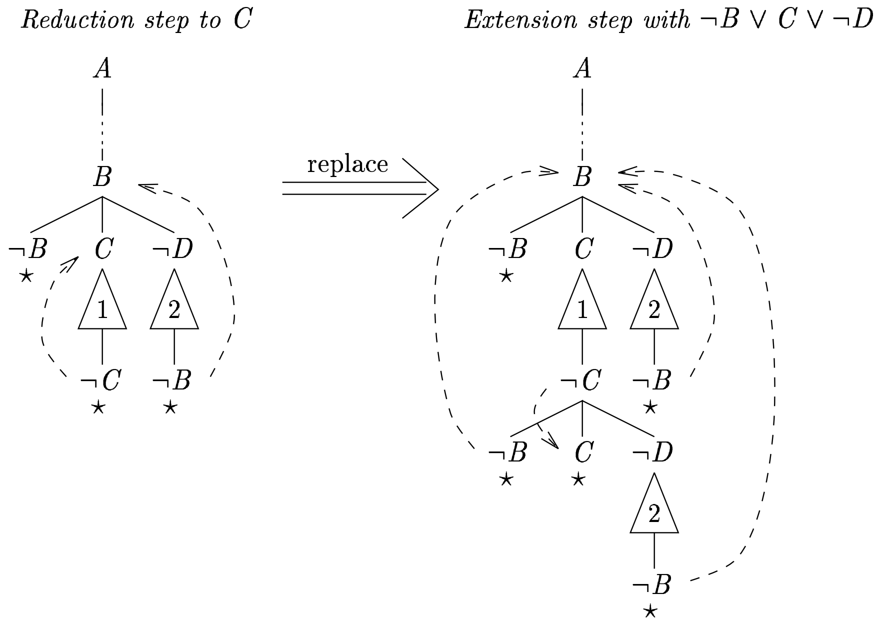


Figure 4: A prototypical example for case 2 in the proof of lemma 3.8: a reduction step occurred to a positive literal which is not topmost.

This concludes the case analysis. Note that in both cases we have obtained $R'_A < R_A$. Let us denote by R' the refutation which is obtained from R by replacing R_A with R'_A. It follows, by definition, that $R' < R$. Hence we can apply the induction hypothesis to R', which concludes the proof of (1).

Ad (2): Having achieved the property (1), (2) is now rather easy. The proof is by induction on the number N of violations of the blockwise regularity restriction in a refutation R without reduction steps. If $N = 0$ we are done, otherwise R contains a tableau with open branch $b = K_1, \ldots, K_m$ which is to be extended in the course of the refutation to an open branch $b' = K_1, \ldots, K_m, \ldots, K_m$. Now, since no reduction steps are applied, we can safely delete from R the subderivation which leads from b to b', and use b instead of b' in the subrefutation of (the open leaf of) b'. This decreases N by one, and we can apply the induction hypothesis. **Q.E.D.**

Before we turn to the proof of the main lemma (Lemma 3.7) we need one more lemma:

Lemma 3.9 *Suppose a ground clause set*

$$\mathcal{S} = \{G,\ A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m\} \cup \mathcal{S}' \qquad (n \geq 2,\ m \geq 0)$$

where G is a ground clause, is minimal unsatisfiable. Then, for some i, $1 \leq i \leq n$, the set

$$\mathcal{S}_i = \{G,\ A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_n \leftarrow B_1,\ldots,B_m\} \cup \mathcal{S}'$$

is unsatisfiable, and furthermore $\mathcal{S}_i \setminus \{G\}$ is satisfiable.

Proof. Clearly, $\mathcal{S}_i$ is unsatisfiable (for every i), because otherwise a model for $\mathcal{S}_i$ would be a model for $\mathcal{S}$.

Let $\mathcal{I}$ be a model for $\mathcal{S} \setminus \{G\}$. Such a model exists because $\mathcal{S}$ is *minimal* unsatisfiable. It holds that $\mathcal{I}$ is a model for $A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$ (because this clause is contained in $\mathcal{S}$). We distinguish two cases:

Case 1: $\mathcal{I}$ is a model for some A_j ($j \in \{1,\ldots,n\}$). But then, $\mathcal{I}$ is a model for $A_1,\ldots,A_{i-1},A_{i+1},\ldots,A_n \leftarrow B_1,\ldots,B_m$, where

$$i := \begin{cases} 2 & \text{if } j = 1 \\ 1 & \text{else} \end{cases}$$

Thus, $\mathcal{I}$ is a model for $\mathcal{S}_i \setminus \{G\}$

Case 2: $\mathcal{I}$ is not a model for some A_j, i.e. A_j is false in $\mathcal{I}$ for all $j = 1,\ldots,n$. Since $\mathcal{I}$ is a model for $A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$ it must be that $\mathcal{I}$ is a model for some body literal B_k. But then, $\mathcal{I}$ is a model for e.g. $A_1,\ldots,A_{n-1} \leftarrow B_1,\ldots,B_m$, i.e. we set $i := n$. Thus, also in this case we have that $\mathcal{I}$ is a model for $\mathcal{S}_i \setminus \{G\}$. **Q.E.D.**

Now for the general case:

Proof. (Lemma 3.7) Some terminology is introduced: if we speak of "replacing a clause C in a derivation by a clause $C \vee D$" we mean the derivation that results when using the clause $C \vee D$ in place of C in extension steps. Also, if $L \in C$ is used as extension literal, then L must be used as extension literal in $C \vee D$ as well.

By a "derivation of a clause C" we mean a derivation that generates a tableau with frontier C.

Let $k(\mathcal{S})$ denote the number of occurrences of positive literals in $\mathcal{S}$ minus the number of definite clauses[8] in $\mathcal{S}$ ($k(\mathcal{S})$ is related to the well-known *excess literal parameter*). Below we will make use of the obvious fact that $\mathcal{S}' \subset \mathcal{S}$ implies $k(\mathcal{S}') \leq k(\mathcal{S})$.

Now we prove the claim by induction on $k(\mathcal{S})$.

Induction start ($k(\mathcal{S}) = 0$): $\mathcal{S}$ must be a set of Horn clauses. Apply Lemma 3.8.

Induction step ($k(\mathcal{S}) > 0$): As the induction hypothesis suppose the result to hold for minimal unsatisfiable ground clause sets $\mathcal{S}'$ with $k(\mathcal{S}') < k(\mathcal{S})$.

Since $k(\mathcal{S}) > 0$, $\mathcal{S}$ must contain a non-Horn clause

$$D = A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$$

with $n \geq 2$. We distinguish two cases:

Case 1. $G \equiv D$, i.e. the selected non-Horn clause is the desired top clause. Let

$$D' = A_1,\ldots,A_{n-1} \leftarrow B_1,\ldots,B_m$$

[8]A *definite clause* is a clause containing exactly one positive literal.

Clearly, $\mathcal{S}_{D'} := (\mathcal{S} \setminus \{D\}) \cup \{D'\}$ is unsatisfiable (because otherwise, a model for $\mathcal{S}_{D'}$ would be a model for $\mathcal{S}$). Furthermore, $\mathcal{S}_{D'} \setminus \{D'\}$ is satisfiable, because otherwise $\mathcal{S}$ would not be *minimal* unsatisfiable. Hence, if we select a minimal unsatisfiable subset $\mathcal{S}'_{D'} \subseteq \mathcal{S}_{D'}$ it must be that $D' \in \mathcal{S}'_{D'}$.

With $k(\mathcal{S}'_{D'}) \leq k(\mathcal{S}_{D'}) = k(\mathcal{S}) - 1 < k(\mathcal{S})$ we can apply the induction hypothesis to $\mathcal{S}'_{D'}$ and obtain a refutation $R_{D'}$ of $\mathcal{S}'_{D'}$ (and thus also of $\mathcal{S}_{D'}$) with top clause D'.

Let $\mathcal{S}_{A_n} := (\mathcal{S} \setminus \{D\}) \cup \{A_n\}$. By the same line of reasoning as for $\mathcal{S}_{D'}$ we can find a minimal unsatisfiable subset $\mathcal{S}'_{A_n} \subseteq \mathcal{S}_{A_n}$, and it must be that $A_n \in \mathcal{S}'_{A_n}$. Similarly, $k(\mathcal{S}'_{A_n}) \leq k(\mathcal{S}_{A_n}) = k(\mathcal{S}) - (n-1) < k(\mathcal{S})$, and we can apply the induction hypothesis to $\mathcal{S}'_{A_n}$ and obtain a refutation R_{A_n} of $\mathcal{S}'_{A_n}$ (and thus also of $\mathcal{S}_{A_n}$) with top clause A_n.

Now replace in $R_{D'}$ every occurrence of the clause D' by D. This gives us a derivation R_D from the input set $\mathcal{S}$. R_D is a derivation of, say, k occurrences of the disjunctive positive literal A_n.

Now append k times the refutation R_{A_n} to R_D in order to obtain subrefutations for every occurrence of A_n in the tableau generated by R_D. Call this new refutation R. Since R_{A_n} might possibly contain extension steps with A_n, R is a refutation of $\mathcal{S} \cup \{A_n\}$. In order to turn R into a refutation of $\mathcal{S}$ alone, replace every extension step with A_n in the appended refutations R_{A_n} by a reduction step to A_n. This is possible, because A_n is the top clause in R_{A_n} and hence can be accessed. Further, these reduction steps are in accordance with the disjunctive positive refinement, because A_n stems from a disjunctive positive clause. The resulting refutation is a desired disjunctive positive ME refutation of $\mathcal{S}$. Furthermore, since $R_{D'}$ and R_{A_n} can be assumed by the induction hypothesis to be blockwise regular, and the construction of the final refutation keeps the block structure of its constituents, the final refutation must be blockwise regular, too.

Case 2: $G \not\equiv D$, i.e. the selected non-Horn clause is different from the desired top clause. According to Lemma 3.9 there exists an appropriate literal A_i $(1 \leq i \leq n)$ to be deleted from D. More formally, we can define

$$D_i = A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \leftarrow B_1, \ldots, B_m$$

in such a way that $\mathcal{S}_{D_i} := (\mathcal{S} \setminus \{D\}) \cup \{D_i\}$ is unsatisfiable and $\mathcal{S}_{D_i} \setminus \{G\}$ is satisfiable. Further, as for $\mathcal{S}_{D'}$ above, $\mathcal{S}_{D_i}$ is unsatisfiable and $\mathcal{S}_{D_i} \setminus \{D_i\}$ is satisfiable. Hence, if we select a minimal unsatisfiable subset $\mathcal{S}'_{D_i} \subseteq \mathcal{S}_{D_i}$ it must be that $G \in \mathcal{S}'_{D_i}$ and $D_i \in \mathcal{S}'_{D_i}$.

With $k(\mathcal{S}'_{D_i}) \leq k(\mathcal{S}_{D_i}) = k(\mathcal{S}) - 1 < k(\mathcal{S})$ we can apply the induction hypothesis to $\mathcal{S}'_{D_i}$ and obtain a refutation R_{D_i} of $\mathcal{S}'_{D_i}$ (and thus also of $\mathcal{S}_{D_i}$) with top clause G.

Let $\mathcal{S}_{A_i} := (\mathcal{S} \setminus \{D\}) \cup \{A_i\}$. As in the previous case for A_n, there exists a refutation R_{A_i} of $\mathcal{S}'_{A_i}$ with top clause A_i, where $\mathcal{S}'_{A_i}$ is defined analogously to $\mathcal{S}'_{A_n}$ above.

The rest of the proof is literally the same as for the previous case, except that D' is replaced by D_i and A_n is replaced by A_i. **Q.E.D.**

4 Backward and Forward Subsumption for Model Elimination

In this section we will refine ME in such a way that it can take advantage of the subsumption deletion techniques which have been applied so successfully in the resolution paradigm.

As described in the introduction, it is crucial in the ME case to define the subsumption relation among tableaux in such a way that as few ancestor literals as possible have to be considered. To assure that subsumption deletion based on such a subsumption relation is complete, a version of ME has to be used which is complete when reduction steps are restricted to just that considered ancestor context. To this end, we defined in Section 3 the "disjunctive positive refinement" of ME.

We will consider both *forward subsumption* and *backward subsumption*. At a superficial level one might think that both of them are quite similar and easily incorporated into any reasonable calculus. While this might be true for resolution systems, it is not for ME, at least if the "usual" definition of refutation is used. Let us explain this now in more detail.

It is more straightforward to incorporate into ME a concept of forward subsumption than backward subsumption. For forward subsumption, the definition of "derivation" (Definition 2.8) can be extended to forbid the derivation of a tableaux T if some prefix of this derivation generates a tableaux T' which subsumes T (wrt. some subsumption relation to be defined). Such an approach has already been considered in [Loveland, 1978].

Usually[9], derivations are constructed in ME proof procedures by nondeterministically guessing the inference steps and backtrack on failure. Forward subsumption is straightforwardly incorporated into such a regime by setting up a failure in case a forward subsumption applies. However, this requires to store tableaux *explicitly* — which is usually not done, for instance, in SETHEO and PTTP based procedures.

Now, backward subsumption is dual to forward subsumption. A first attempt to define backward subsumption within the given framework is to say that tableau T backward subsumes a tableau T' in a derivation D iff T (strictly) subsumes T' and T' was generated in D prior to T. The information which can be gained from a concrete backward subsumption case can be expressed as follows: "if T cannot be extended to a refutation, then also T' cannot be extended to a refutation". In terms of a proof procedure this means that failure to prove T should immediately backtrack to the tableau before T' and explore an alternative to T'. In addition to explicitly store tableaux, this requires to manipulate the backtracking scheme in depth.

We will not elaborate on the practicability of such an approach. Instead we will return to the calculus level and propose a new notion of derivation which supports our needs. More precisely, instead of guessing the "next" tableau we will explicitly generate all possible successor tableaux. By this we change ME from an enumeration procedure of *derivations* into a saturation procedure for *formulas*. This is the approach of resolution, and, as there, it allows us to conveniently express backward subsumption as a simply deletion operation.

A general subsumption concept which is related to ours is defined in [Letz *et al.*,

[9]For instance, in SETHEO and the whole class of PTTP theorem provers.

1994]. As we do, they consider explicitly generated tableaux. Unlike our approach, which enumerates these tableaux sequentially, they generate AND/OR search trees whose nodes are labeled with tableaux. They define a notion of subsumption deletion which allows to give up a tableau if it is (not necessarily strictly) subsumed by a competitive tableau in the search tree. Our approach differs from that wrt. the following aspects: for the first, they do not restrict the ancestor context for the subsumption test; they even do not consider permutations for this. Thus, subsumption will apply only rarely. For the second, as they note by themselves, their approach is incomplete. Our work can be seen to solve open issues in their work by identifying several technicalities which are needed to obtain a complete calculus. In our terminology, their calculus is incomplete at least for the reason that they do not use the *strict* version of the subsumption test for backward subsumption.

4.1 Definition of Model Elimination with Subsumption

Definition 4.1 (Subsumption) Let T be a tableau and $b = N_1, \ldots, N_n$ be a branch of T. Define the *disjunctive positive ancestor context of* b, $anc^+(b)$, as

$$anc^+(b) = \{L_i \mid N_i \text{ is a disjunctive positive node, labeled with } L_i,\ i = 1, \ldots, n\}$$

Thus we simply collect the disjunctive positive literals.

Let b_1 and b_2 be branches of tableaux T_1 and T_2, respectively. We say that b_1 is an *ancestor extension* of b_2, $b_1 \geq b_2$, iff

$$leaf(b_1) = leaf(b_2) \text{ and } anc^+(b_1) \supseteq anc^+(b_2)$$

In words, b_1 is an ancestor extension of b_2 iff the leaves are equal and b_1 has at least the disjunctive positive ancestor context of b_2.

Extending to branch sets, we say that a branch set B_1 is an *ancestor extension* of a branch set B_2, $B_1 \geq B_2$, iff there exists a bijective function f from B_1 into B_2 such that $b \geq f(b)$ for every $b \in B_1$. Thus, in particular, subsumption is based on comparing *multi*sets of goal literals.

We intend to generalize towards tableaux. In the sequel let T, T_1, T_2 denote tableaux and let δ denote a substitution.

We say that T_1 *is more general by* δ than for T_2 iff[10] $OB(T_1\delta) \geq OB(T_2)$; T_1 is *more general* than T_2 iff T_1 is more general than T_2 by some substitution δ. Note that this relation is reflexive and transitive, but not antisymmetric, and hence a preorder.

We say that T_1 *strictly* δ*-subsumes* T_2, $T_1 \lhd_\delta T_2$ iff $OB(T_1\delta) \geq B_2$ for some subset $B_2 \subset OB(T_2)$. Furthermore, T_1 *strictly subsumes* T_2, $T_1 \lhd T_2$, iff there exists a substitution δ such that $T_1 \lhd_\delta T_2$. In words, T_1 strictly subsumes T_2 if by some substitution the branches of T_1 are an ancestor extension of some of the branches of T_2. We note that as a consequence of the well-foundedness of strict multiset inclusion, strict subsumption is also well-founded.

Finally, we say that T_1 *subsumes* T_2, $T_1 \unlhd T_2$, iff $T_1 \lhd T_2$ or T_1 is more general than T_2. Note that this is a slight abuse of notation as the "more general" relation is not an equivalence relation, not even if no substitutions are involved. **(End Definition)**

[10] "OB" is defined in Definition 2.3.

Example 4.2 Figure 5 depicts some tableaux and subsumption relations among them. **(End Example)**

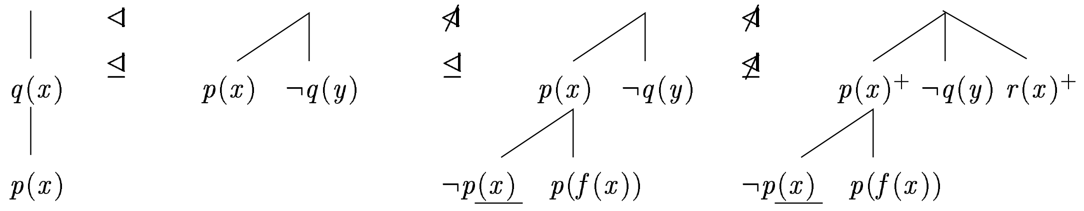


Figure 5: Subsumption relations among some tableaux.

Definition 4.3 (Model Elimination with Subsumption (MES)) Let r be a computation rule. For brevity of notation we define $T \vdash_{\textbf{Infer},r} T'$ iff the tableau T' can be obtained from the tableau T by one single application of one of the inference rule *extension step* or *reduction step* of the disjunctive positive refinement (Definition 3.2) to the goal node selected by r in T.

We define the binary relations "$\vdash_{\textbf{Infer},r}$" and "$\vdash_{\textbf{Delete}}$" on sets of tableaux as follows: $S \vdash_{\textbf{Infer},r} S'$ (resp. $S \vdash_{\textbf{Delete}} S'$) if S' is obtained from S by one single application of the following respective inference rules (also called *derivation rules* from now on):

Infer: $$\frac{S \cup \{T\}}{S \cup \{T,\ T'\}} \quad \Big\{ \text{ If } T \vdash_{\textbf{Infer},r} T'$$

Delete: $$\frac{S \cup \{T,\ T'\}}{S \cup \{T\}} \quad \Big\{ \text{ If } T \lhd T'$$

The calculus *forward and backward subsumption ME, (MES)* consists of the derivation rules **Infer** and **Delete**; the calculus *forward subsumption ME* consists of the derivation rule **Infer** only.

A *derivation* (wrt. one of these calculi) for a clause set $\mathcal{S}$, goal set $\mathcal{G} \subseteq \mathcal{S}$ and computation rule r is a sequence

$$S = (S_0, S_1, \ldots, S_n, \ldots)$$

of sets of tableaux of $\mathcal{S}$, where

1. $S_0 = \{T_G \mid G \in \mathcal{G},\ T_G \text{ is an initial tableau for } G\}$, and

2. for $i > 0$, either $S_{i-1} \vdash_{\textbf{Infer},r} S_i$ or $S_{i-1} \vdash_{\textbf{Delete}} S_i$ ("Backward subsumption" — only for the forward and backward variant).

Finally, a *refutation* of a clause set $\mathcal{S}$ is a derivation for $\mathcal{S}$ such that one of its elements contains a closed tableau. **(End Definition)**

In words: beginning with a set of initial tableaux, we generate new tableaux by means of the traditional inference rules, and we also allow to delete a previously derived tableau, provided that it is strictly subsumed in the current proof state. By this *backward* subsumption deletion we avoid further exploration of the subsumed tableau. It is important to use the *strict* version of subsumption here in order to avoid infinite loops. *Forward* subsumption is covered by a certain property of fair derivations (cf. Definition 4.4 below). Informally, it says that a tableau need not be generated in presence of a subsuming tableau.

In the traditional definition of *derivation* (Def. 2.8) the transition from one tableau to the next can be thought of as a nondeterministical guess. In the new framework we will use a *fairness* condition instead. Roughly, fairness means that no application of an **Infer** derivation rule is deferred infinitely long. Fairness is important since it entails that "enough" tableaux will be generated; in particular a closed tableau will be generated after finitely many steps for unsatisfiable clause sets.

Our definition of fairness is an adaption of standard definitions in the term-rewriting literature (see e.g. [Bachmair, 1991]).

Definition 4.4 (Limit, Fairness) Let $S = (S_0, S_1, \ldots, S_n, \ldots)$ be a derivation for some clause set $\mathcal{S}$. The *limit* of S is defined as

$$S_\infty := \bigcup_{(i \geq 0)} \bigcap_{(j \geq i)} S_j$$

The elements of S_∞ are also called the *persisting* tableaux of S.

The derivation S is called *fair* iff S is either a refutation, or else whenever $S_\infty \vdash_{\mathbf{Infer},r} S_\infty \cup \{T'\}$ then for some $k \geq 0$ and some $T \in S_k$ we have $T \trianglelefteq T'$. **(End Definition)**

The first item in the definition of derivation above (Definition 4.3) guarantees in conjunction with fairness that every clause given from the goal set $\mathcal{G}$ is tried as a top clause; when instantiated with all negative clauses from $\mathcal{S}$ this corresponds to the result for "traditional" ME which states that if a refutation exists at all, then also a refutation with negative clause as top clause exists.

The fairness condition, proper, means that it is sufficient to generate a new tableau from the persisting tableaux[11] only if it is not subsumed in some stage of the derivation (provided S is not a refutation). Since this case includes the possibility of discarding a tableau in favor of a subsuming *previously* derived tableau, we have a case of *forward* subsumption. Note further that we need not insist on *strict* subsumption here.

The presence of a computation rule is most important in the fairness condition. It is sufficient to consider inferences to the *selected* branch only, but not to all branches of a tableau.

Our notion of fairness enables the use of a "delete as many tableaux as possible" strategy in implementations, since a tableau once shown to be subsumed will be subsumed in all subsequent stages and thus need not persist.

[11]Informally, these are the tableaux generated eventually and never deleted afterwards.

Example 4.5 (1) Consider the clause set

$$\begin{array}{llll} (1) & p(a) & \leftarrow & p(y) \\ (2) & p(a) & \leftarrow & r(z) \\ (3) & r(c) & \leftarrow & \\ (4) & & \leftarrow & p(x), q(x) \end{array}$$

Suppose a computation rule which selects the bottom-most leftmost goal. It follows a fair derivation for the goal clause (4). Since this example is Horn, it suffices to depict the frontiers of the tableaux as clauses instead of the tableaux themselves. The clause sets in parenthesis need not be built as the generated tableau in these steps are subsumed. The selected literal in each step is underlined.

$S_0 = \{\underline{\neg p(x)} \vee \neg q(x)\}$		Initial tableau with (4)
$S_1 = \{\neg p(x) \vee \neg q(x),$ $\underline{\neg p(y)} \vee \neg q(a)\}$		By extension with (1)
$(S_2 = \{\neg p(x) \vee \neg q(x),$ $\neg p(y) \vee \neg q(a),$ $\neg p(y) \vee \neg q(a)\})$		By extension with (1)
$S_2 = \{\neg p(x) \vee \neg q(x),$ $\neg p(y) \vee \neg q(a),$ $\underline{\neg r(z)} \vee \neg q(a)\}$		By extension with (2)
$S_3 = \{\neg p(x) \vee \neg q(x),$ $\neg p(y) \vee \neg q(a),$ $\neg q(a)\}$		By extension with (3)
$S_4 = \{\underline{\neg p(x)} \vee \neg q(x),$ $\neg q(a)\}$		By deletion of $\neg p(y) \vee \neg q(a)$
$(S_5 = \{\neg p(x) \vee \neg q(x),$ $\neg q(a),$ $\neg p(y) \vee \neg q(a)\})$		By extension with (1). Unnecessary, because the new tableau is subsumed by $\neg p(y) \vee \neg q(a) \in S_1$
$(S_5 = \{\neg p(x) \vee \neg q(x),$ $\neg q(a),$ $\neg r(z) \vee \neg q(a)\})$		By extension with (2). Unnecessary, because the new tableau is subsumed by $\neg r(z) \vee \neg q(a) \in S_2$

Thus, the derivation is finite and ends in S_4.

(2) Consider the clause set

$$\begin{array}{lll}
(1) & x = x & \leftarrow \\
(2) & y = x & \leftarrow \quad x = y \\
(3) & x = z & \leftarrow \quad x = y,\ y = z \\
(4) & f(x) = f(y) & \leftarrow \quad x = y \\
(5) & p(x, y) & \leftarrow \quad x = x',\ p(x', y) \\
(6) & p(x, y) & \leftarrow \quad y = y',\ p(x, y') \\
(7) & & \leftarrow \quad x = f(y),\ p(z, y)
\end{array}$$

Of course, clauses (1)-(6) are an axiom set for the theory of equality in the presence of a predicate symbol p and a function symbol f. Sure, there are better ways to handle equality, but this example should demonstrate that part of the improvements achieved for inference rules for equality, notably *paramodulation* [Robinson and Wos, 1969], can be obtained as instances of subsumption.

When started with the top clause (7) and equipped with the same computation rule as above, our subsumption prover stops after 9 inferences and (correctly) reports failure, while other provers (PROTEIN – a PTTP prover, and SETHEO) loop forever on this example.

We will only develop some key inferences of the derivation. Let it start with the tableau[12] $\neg(x = f(y)) \vee \neg p(z, y)$ for (7). Extension with (4) yields a tableau $\neg(x = f(y)) \vee \neg p(z, y)$ which is immediately (forward) subsumed. More generally, a frontier of one of the forms

$$\begin{array}{l}
\neg(x_1 = y_1) \vee R \\
\neg(x_1 = f(y_1, \ldots, y_n) \vee R \\
\neg(f(x_1, \ldots, x_n) = y_1) \vee R \\
\neg(f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)) \vee R
\end{array}$$

where R is a clause which does not contain one of the variables $\{x_1, \ldots, x_n, y_1, \ldots y_n\}$ need never be subject to an extension step with a functional reflexive axiom (4) because the resulting tableau would be subsumed. Of course, this holds only if the ancestor context is in accordance.

This is an important restriction because it avoids many infinite branches. This restriction has an interesting interpretation from the viewpoint of "paramodulation". It is well-known that goal-oriented calculi, such as ME or linear Resolution system [Furbach *et al.*, 1989], require to paramodulate into and even below variables when equipped with a paramodulation inference rule. Now, the purpose of the functional reflexive axioms is to simulate paramodulation into and below variables. Together with our considerations above, this suggests that paramodulation into and below variables can be avoided under the stated conditions.

Let us now continue the example. Extension of $\neg(x = f(y)) \vee \neg p(z, y)$ with (1) yields $\neg p(z, y)$ which backward subsumes the initial tableau. Extension of $\neg p(z, y)$ with (5) yields $\neg p(x', y) \vee \neg(x' = z)$, which is extended by (1) to $\neg p(z, y)$. Here forward subsumption applies again. Intuitively, paramodulation (by means of the subsumption replacement axiom) into the variables of $\neg p(z, y)$ is redundant. **(End Example)**

[12] Again, since we deal with a Horn example we write clauses instead of tableaux.

Subsumption is a concept strong enough to to generalize some well-known refinements of ME. For instance, *blockwise regularity* is covered by forward and backward subsumption in the case of a local computation rule (cf. Theorem 5.13 below).

Another well-known refinement of ME is *factorization* [Loveland, 1978]. The factorization inference rule allows to close a branch b iff $\mathit{leaf}(b)$ is identical to a left brother node of some ancestor node of $\mathit{leaf}(b)$. It is allowed to apply some (most general) substitution to achieve this condition. An important special case is that the two literals in question are identical without application of a substitution[13]. In ME proof procedures, often a folklore-like heuristic allows to close the branch b without need for backtracking. Backward subsumption explains why this step is correct: suppose a tableau T' is obtained from a tableau T by factorization with empty substitution. Then it is easy to see that $T' \lhd T$. Hence T can be deleted in a fair derivation without affecting completeness; this corresponds to non-backtracking over this inference step in backtracking procedures.

4.2 Soundness and Completeness

Since ordinary ME is sound, the disjunctive positive refinement of the previous section must be sound, too (because the latter case is more restricted). But then, MES must be sound, too, because it is more restricted (i.e. whenever a tableau can be constructed in MES, it can also be constructed in the disjunctive positive refinement). Altogether, this sketch convinces us of the soundness of the calculus. We will thus turn towards completeness now.

Completeness reads as follows:

Theorem 4.6 (Completeness) *Let r be a computation rule. Let $\mathcal{S}$ be an unsatisfiable clause set, and let*

$$S = (S_0, S_1, \ldots, S_n, \ldots)$$

be a fair MES derivation for $\mathcal{S}$ with goal clause set $\mathcal{G}$, where $\mathcal{G}$ is the set of negative clauses from $\mathcal{S}$. Then for some $q > 0$, S_q contains a closed tableau, i.e. S is a refutation.

Before we can prove this theorem we find it helpful to introduce some notation and lemmas. Unless otherwise noted, we will for the rest of this section always refer to the disjunctive positive refinement of ME (Definition 3.2), and r will always denote a given computation rule.

Let $D = d_0, \ldots, d_m$ be a derivation via r generating a tableau T_n. We call D *promising* iff D can be extended to a refutation, i.e. iff a refutation
$d_0, \ldots, d_m, d_{m+1}, \ldots, d_n$ via r exists, for some $n \geq m$. In this case we shall restrict among all possible refutations to those of minimal length. Hence suppose without loss of generality n to be minimal, and define $\mathit{steps}(D) := n - m$ as the minimal number of inference steps necessary to extend D to a refutation; obviously, $\mathit{steps}(D) = 0$ iff D is a refutation. The value $\mathit{steps}(D)$ will be of central importance below, since it is the measure to be decreased in the induction step. It is important to have that $\mathit{steps}(D)$ does not depend from the computation rule r. Indeed, this independence

[13]This option is included in Loveland's original definition of ME [Loveland, 1978].

is a consequence of the proof of the independence of the computation rule. For this proof (cf. [Baumgartner, 1994]) it requires only to switch inference steps of a given derivations, but never to delete or add some.

The previous definitions shall also be applicable to T_m, and in this case it is understood that T_m is generated by some appropriate derivation which is left implicit.

Now let S be a set of tableaux and T be a tableau. We say that S *is minimal for* T iff T is promising and for every promising tableau $T' \in S$ it holds $steps(T) \leq steps(T')$. Note that according to this definition $T \in S$ does not necessarily hold.

We start with two lemmas:

Lemma 4.7 *Let T' be a promising tableau, and let D be a derivation generating a tableau T which is more general than T'. Then T is promising, and furthermore $steps(T) \leq steps(T')$.*

Proof. In the sequel if we speak of a "subrefutation of a tableau T" we mean the concatenation of the subrefutations of the nodes of the frontier of T.

We are given that T' is promising, which means that there exists a subrefutation of T'.

By definition of "more general" we find a substitution δ such that for every open branch in T' there is exactly one open branch in $T\delta$ (and these are all the open branches of $T\delta$). Furthermore, two corresponding branches $b \in T\delta$ and $b' \in T'$ have the same leaf literals, and, $anc^+(b) \supseteq anc^+(b')$. Hence the existing subrefutation of the leaf of b' can also be applied to the leaf of b. If this is done for every such branch $b' \in T'$ we finally arrive at a subrefutation of $T\delta$, and it holds $steps(T\delta) = steps(T')$. However, since $anc^+(b) \supset anc^+(b')$ might hold in some cases, reduction steps could be enabled in the refutation of $T\delta$ which have no counterpart in T'. Hence even $steps(T\delta) < steps(T')$ is possible.

We have to show the claim for T, but not for $T\delta$. This can be done by lifting the subrefutation of $T\delta$ to a subrefutation of T. For this we need a lifting theorem for ME derivations. There exist several such results; the lifting theorem needed in our case can be proven with methods developed in [Baumgartner *et al.*, 1995]. Notably, lifting does not change the number of inference steps which gives us $steps(T) \leq steps(T')$. **Q.E.D.**

Lemma 4.8 *Let T' be a promising open tableau, and let D be a derivation generating a tableau T with $T \lhd T'$. Then T is promising, and furthermore $steps(T) < steps(T')$.*

Proof. The proof is very similar to the one of the previous lemma. The major difference is that we now have a one-to-one correspondence between the open branches B of $T\delta$ and a *strict* subset B' of the open branches of T'. Thus we can apply the subrefutations of the leafs of the branches of B' to their corresponding branches B. This gives us a subrefutation of $T\delta$. Again using a lifting theorem we obtain a subrefutation of T using the same number of inference steps. Since T' contains strictly more open branches than T, the subrefutation of T' needs strictly more inference steps than that of T, or, equivalently, $steps(T) < steps(T')$. **Q.E.D.**

The next lemma is central:

Lemma 4.9 *Let $S = (S_0, S_1, \ldots, S_n, \ldots)$ be a fair derivation which is not a refutation (i.e. the empty tableaux is never generated). If S_k is minimal for T_k and $steps(T_k) > 0$ then there exists an index $l \geq 0$ and a tableau $T_l \in S_l$ such that S_l is minimal for T_l and $steps(T_l) < steps(T_k)$.*

Proof. By definition of minimality, T_k is promising. Since $steps(T_k) > 0$ at least one inference step has to be applied to extend T_k to a refutation. Let

$$T_k \vdash_{\mathbf{Infer}} T' \tag{1}$$

be such an inference step. It holds $steps(T') = steps(T_k) - 1 < steps(T_k)$ (*).

Now we distinguish two cases:

1. $T_k \in \bigcup_{(i \geq k)} \bigcap_{(j \geq i)} S_j$, i.e. T_k is generated at some point $\geq k$ and then never deleted afterwards. But then also

$$T_k \in \bigcup_{(i \geq 0)} \bigcap_{(j \geq i)} S_j \quad (= S_\infty)$$

because this is a superset. From this and Equation 1 it follows $S_\infty \vdash_{\mathbf{Infer},r} S_\infty \cup \{T'\}$. But then, by fairness, there exists an index $l \geq 0$ and tableau $T \in S_l$ such that (a) $T \lhd T'$ or (b) T is more general than T'. In case (a), if T is closed we have $steps(T) = 0$ $(< steps(T_k))$ or otherwise T is open. But then, by Lemma 4.8, T can be extended to a refutation such that $steps(T) < steps(T')$, and together with (*) above we conclude $steps(T) < steps(T_k)$. In case (b) we apply Lemma 4.7 and conclude that T can be extended to a refutation such that $steps(T) \leq steps(T')$, which also gives us $steps(T) < steps(T_k)$.

2. $T_k \notin \bigcup_{(i \geq k)} \bigcap_{(j \geq i)} S_j$, i.e. for some $l \geq k+1$ we have $T_k \in S_{l-1}$ and $T_k \notin S_l$. In other words, T_k is deleted. This can only be done by backward subsumption pruning, i.e.

$$(S_{l-1} =)\ S_l \cup \{T_k\} \vdash_{\mathbf{Delete}} S_l$$

By definition of **Delete** there exists a tableau $T \in S_l$ with $T \lhd T_k$. By Lemma 4.8, T can be extended to a refutation such that $steps(T) < steps(T_k)$. This concludes the second case.

Now, in both cases, we have shown that there exists a tableau $T \in S_l$ such that $steps(T) < steps(T_k)$. Either S_l is already minimal for T (and in this case define $T_l := T$), or otherwise S_l is minimal for some different $T_l \in S_l$ (and in this case even $steps(T_l) < steps(T)$ holds). Thus, in any case we can find a tableau T_l as claimed. **Q.E.D.**

Now we can proof our main theorem of above:

Proof. (Theorem 4.6). By the completeness theorem for the disjunctive positive refinement (Theorem 3.6) there exists a refutation of $\mathcal{S}$ with some negative clause $G \in \mathcal{S}$ as top clause. Without loss of generality assume that among all negative clauses for which a refutation exists, G is chosen minimal wrt. the length of the shortest possible refutation.

We are given that the goal clause set $\mathcal{G}$ used for S consists of all negative clauses from $\mathcal{S}$. Hence, by definition of derivation also $T_G \in S_0$, where T_G is an initial

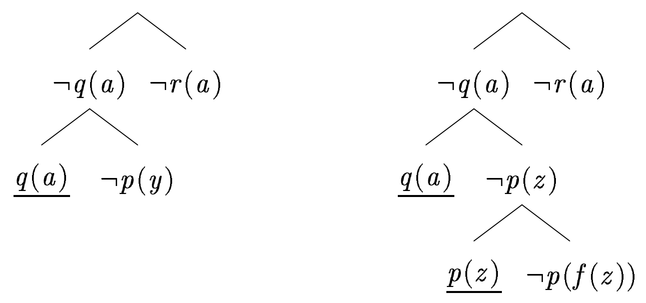

$\neg q(a)$ $\neg r(a)$
$\underline{q(a)}$ $\neg p(y)$
$\neg q(a)$ $\neg r(a)$
$\underline{q(a)}$ $\neg p(z)$
$\underline{p(z)}$ $\neg p(f(z))$

Figure 6: Tableaux for Example 5.1

tableau for G. Further, by the choice of G we have in present terminology that T_G is promising, $T_G > 0$, and S_0 is minimal for T_G.

Now suppose, to the contrary of the theorem, that S is not a refutation. But then, by repeatedly applying Lemma 4.9, starting with S_0 and $T_0 = T_G$ we can find an *infinite* chain $steps(T_0) > steps(T_1) > \ldots > steps(T_n) > \ldots > 0$ which is impossible. Hence, $\mathcal{S}$ is the refutation as claimed. **Q.E.D.**

5 A Context Check for Model Elimination

An interesting alternative to the techniques presented in the last section are so-called *context checks* which were originally proposed in [Besnard, 1989] and further developed in [Bol *et al.*, 1991] in the context of SLD-resolution. Context checks are based on forward subsumption; but instead of comparing two multisets of open goals, the idea is to prune a derivation if a subrefutation for only one open goal would constitute a subrefutation for one of its ancestor goals. Hence, such pruning techniques can be interpreted as *loop checking* mechanisms.

Clearly, in order to prune a derivation, one cannot concentrate on one open goal and its ancestors. Additionally, variable dependencies with other open goals have to be taken into account. For illustration consider the following example.

Example 5.1

$$\begin{array}{lrcl}
(1) & r(a) & \leftarrow & \\
(2) & p(f^5(b)), q(a) & \leftarrow & \\
(3) & p(z) & \leftarrow & p(f(z)) \\
(4) & q(a) & \leftarrow & p(y) \\
(5) & & \leftarrow & q(x), r(x)
\end{array}$$

The task is to find a ME refutation of the (non-Horn) clause set with top-clause (5). After an initialization and one extension step with clause (4) we get the tableau depicted in Figure 6 on the left. One further extension step applied to the open goal $\neg p(y)$ results in the tableau shown on the right. Now, it is quite easy to see, that the last extension step is redundant: Every subrefutation R for $\neg p(f(z))$ immediately constitutes a subrefutation for $\neg p(y)$ (via an additional substitution) $\{y\backslash f(z)\}$. This is because due to the positive refinement, we know that none of the ancestors of $\neg p(f(z))$ is needed for the application of reduction steps. Further, since the variable y does not occur in other open goals, the application of R to $\neg p(y)$ does not affect the possibility to find an overall refutation.[14] **(End Example)**

In order to define a loop check which shows the desired behavior we first have to explain what it means to *undo derivation steps*. Intuitively speaking, the derivation steps applied to a node N (and its successors) are undone by cutting away all successor nodes of N (and removing the substitutions that are due to the corresponding derivation steps.)

Definition 5.2 Let $D = d_1, \ldots, d_n$ be a derivation generating a tableau T, and N be the root node of an open branch in T. Then, let

$$D' = d'_{l_1}, \ldots, d'_{r_1}, d'_{l_2}, \ldots, d'_{r_2}, \quad \ldots \quad , d'_{l_k}, \ldots, d'_{r_k}$$

be a derivation such that

- $1 \leq l_1 \leq r_1 < l_2 \leq r_2 < \ldots < l_k \leq r_k \leq n$,
- the sequence $d_{l_1}, \ldots, d_{r_1}, d_{l_2}, \ldots, d_{r_2}, \quad \ldots \quad , d_{l_k}, \ldots, d_{r_k}$ contains all elements of D but the derivation steps applied to N (and its descendants), and
- for each $j \in \bigcup_{i=1}^{k}\{l_i, \ldots, r_i\}$, d'_j equals d_j except that d'_j might use a different MGU.

We call the tableau T' generated by D' the *tableau emerging from T by undoing the derivation steps applied to N and its descendants.* **(End Definition)**

Note 5.3 For derivations using a *local* computation rule (see Definition 2.10), T' is the tableau generated after d_i for some $i < n$. This is because such a computation rule, after having selected a node N to apply a derivation step, does not select another node occurring in the same successor sequence until each branch containing N is closed. In view of an implementation this is obviously useful: In the course of a deduction, one merely has to memorize the sequence S of tableaux generated during the current derivation. Then, the tableau that emerges by undoing the derivation steps applied to some node N is simply the element of the S which was modified by a derivation step applied to N. **(End Note)**

With this definition we are now able to introduce the so-called T-Context Check.[15] Following [Bol *et al.*, 1991], where loop checks are defined as sets of derivations fulfilling certain criteria[16], we define the T-Context Check as the set of all ME derivations

[14]Clearly, the use of forward-subsumption (as defined previously), also prunes this derivation. However, there are examples which show that none of these "variants" of forward-subsumption is strictly stronger than the other.

[15]The T-Context Check constitutes a proper generalization of the Context Check proposed in [Besnard, 1989] which is, in case Horn clause sets are considered, a generalization of the Identical Ancestor Check.

[16]More formally, a *loop check* is a computable set $\mathcal{L}$ of connection tableau derivations such that for every derivation $D \in \mathcal{L}$ each variant of D is contained in $\mathcal{L}$, too (i.e. $\mathcal{L}$ is closed under variants).

which can be pruned because, roughly speaking, a goal is subsumed by one of its ancestor goals. Clearly, as discussed above, we have to take care of disjunctive positive ancestors. Further, variable dependencies between the goals under consideration and other open goals and their disjunctive positive ancestors have to be taken into account.

Definition 5.4 (T-Context Check) Let D be a ME derivation generating a tableau T which contains an open branch b with leaf node N_l. Let N_a be an ancestor of N_l such that neither N_l nor N_a or any of its descendants which are ancestors of N_l is disjunctive positive. Further, let T' be the tableau emerging from T by undoing all derivation steps applied to N_a and its descendants, let $S' = \{G_1, \ldots, G_m, N_a\}$ be the frontier of T', and let $A_1, \ldots, A_k$ be the disjunctive positive ancestors of nodes in S'.

D belongs to the *Tableau Context Check* (T-Context Check for short) iff there exists a substitution θ such that $L(A_i, T')\theta = L(A_i, T)$ for all $i \in \{1, \ldots, k\}$, $L(G_i, T')\theta = L(G_i, T)$ for all $i \in \{1, \ldots, m\}$, and $L(N_a, T')\theta = L(N_l, T)$. **(End Definition)**

Example 5.5 We continue Example 5.1. In terms of Definition 5.4, let T be the tableau shown in Figure 6 on the right hand side. Further, let N_a and N_l be the nodes in T labeled with the literals $\neg p(z)$ and $\neg p(f(z))$, respectively. Then, the tableau T' which emerges from T by undoing all derivation steps applied to N_a and its successors is the tableau depicted in Figure 6 on the left. We get $k = 0$, $m = 1$, $L(G_1, T') = \neg r(a)$, $L(N_a, T') = \neg p(y)$, $L(G_1, T) = \neg r(a)$, and $L(N_l, T) = \neg p(f(z))$.

Now we can conclude that the derivation generating T belongs to the T-Context Check because no node in T is disjunctive positive and for $\theta = \{y \backslash f(z)\}$ we have $L(G_1, T')\theta = L(G_1, T)$ and $L(N_a, T')\theta = L(N_l, T)$. **(End Example)**

We now have to prove that the T-Context Check preserves completeness. Importantly this holds in combination with blockwise regularity.

Theorem 5.6 *The completeness of blockwise regular ME is preserved even if each derivation belonging to the T-Context Check is pruned.*

Proof. Let D be a derivation belonging to the T-Context Check. If D cannot be completed to a refutation by a sequence $D' = d_1, \ldots, d_n$ of inference steps, it is obviously correct to prune D. Otherwise we have to prove that there is a tableau which is smaller than T (i.e. that is it can be generated with less derivation steps than T) and can be extended to a closed tableau by a derivation of length n' where $n' \leq n$. Since this closed tableau can be built using an arbitrary computation rule (see Proposition 2.11) we do not have to take particular computation rules into account.

We can assume, due to the disjunctive positive refinement of ME, that reduction steps in D' neither use the literal attached to N_l nor the literals attached to N_a or its successors which are ancestors of N_l. Let σ be the composition of substitutions needed for D' and let $d_{k_1}, \ldots, d_{k_r}$ ($1 \leq k_1 < k_2 < \ldots < k_r \leq n$) be the inference steps in D' which are applied to elements of $S' - \{N_a\} \cup \{N_l\}$. Furthermore, let $d'_{k_1}, \ldots, d'_{k_r}$ be an (unrestricted) sequence of derivation steps where d'_{k_1} equals d_{k_1} but uses the substitution $\theta\sigma$ and for $i \in \{2, \ldots, r\}$, d'_{k_i} equals d_{k_i} but uses the substitution σ.

Since $L(A_i, T')\theta = L(A_i, T)$ for all $i \in \{1, \ldots, k\}$ and $L(G_i, T')\theta = L(G_i, T)$ for all $i \in \{1, \ldots, m\}$, it is guaranteed that (i) for each open goal attached to a node $N \in S' - \{N_a\}$, $L(N, T'\theta) = L(N, T)$ and that (ii) for each disjunctive positive ancestor N of an open goal in $T'\theta$, $L(N, T'\theta) = L(N, T)$. Since, furthermore, $L(N_a, T')\theta = L(N_l, T)$ holds, it is easy to see that the unrestricted derivation $d'_{k_1}, \ldots, d'_{k_r}$ closes the tableau T' (which obviously can be build with less derivation steps than T). But then there also exists a sequence of inference steps $d''_{k_1}, \ldots, d''_{k_r}$ which closes T' (note that $r \leq n$) where for $i \in \{1, \ldots, r\}$, d''_{k_i} equals d'_{k_i} but uses a more general substitution (this can be proven via a lemma which is similar to Lemma 8.1 in [Lloyd, 1987]; for instance see [Brüning, 1994]).

It is however easy to recognize that the tableau T_1 generated by $D, d''_{k_1}, \ldots, d''_{k_r}$ might violate blockwise regularity. But this does not impose any problems: Suppose some open branch b in T_1 violates blockwise regularity because some node K_1 and its descendant K_2 occurring in the same block of b are labeled with the same literal. Let ρ be the composition of substitutions needed to apply the derivation steps in $D, d''_{k_1}, \ldots, d''_{k_r}$. Then it is obviously possible to apply an unrestricted subrefutation to K_1 which equals the subrefutation for K_2 except that each of the involved derivation steps uses the substitution ρ. Let T_2 be the resulting tableau. With the same argumentation as above we can infer that there exist a (restricted) refutation which generates a tableau T_3 such that T_2 is an instance of T_3. **Q.E.D.**

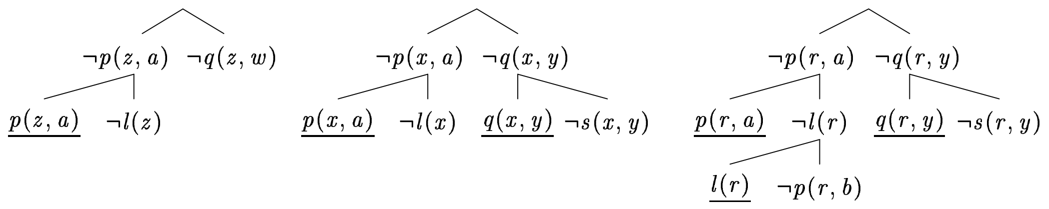
Note 5.7 The T-Context Check is compatible with disjunctive regularity, too (note, however, that we do not know whether disjunctive regular ME is complete). To prove this, we assume that (speaking in terms used in the above proof) D, D' does not violate disjunctive regularity. Then it is easy to see that $D, d'_{k_1}, \ldots, d'_{k_r}$ cannot violate disjunctive regularity, too, since the latter (unrestricted) derivation generates a tableau which equals the one generated by D, D' except that some branches are "shortened" (recall that d'_{k_1} uses the substitution $\theta\sigma$). But then, $D'' = D, d''_{k_1}, \ldots, d''_{k_r}$ cannot violate disjunctive regularity, too, since the tableau generated by $D, d'_{k_1}, \ldots, d'_{k_r}$ is an instance of the tableau generated by D''. **(End Note)**

An interesting aspect of mechanisms for avoiding redundancies is their (in-)dependence of the employed search strategy (see also [Brüning, 1994]). In what follows we shall illustrate that the success of forward subsumption in fact depends on this relationship whereas the T-Context Check prunes a derivation regardless of the order in which the involved derivation steps are performed.

Example 5.8

$$\begin{array}{llcl}
(1) & q(x,y) & \leftarrow & s(x,y) \\
(2) & p(z,a) & \leftarrow & l(z) \\
(3) & l(r) & \leftarrow & p(r,b) \\
(4) & & \leftarrow & p(u,v), q(u,w)
\end{array}$$

A ME derivation of this clause set is depicted in Figure 7. First, an initialization step is performed with top-clause (4). Afterwards, extension steps are applied with



$\neg p(z,a)$ $\neg q(z,w)$
$p(z,a)$ $\neg l(z)$
$\neg p(x,a)$ $\neg q(x,y)$
$p(x,a)$ $\neg l(x)$ $q(x,y)$ $\neg s(x,y)$
$\neg p(r,a)$ $\neg q(r,y)$
$p(r,a)$ $\neg l(r)$ $q(r,y)$ $\neg s(r,y)$
$l(r)$ $\neg p(r,b)$

Figure 7: Tableaux for Example 5.8

clause (2), clause (1), and clause (3), respectively. This derivation is pruned by the T-Context Check: Let N_l be the node labeled with $\neg p(r,b)$ and let N_a be its ancestor labeled with $\neg p(r,a)$. In Terms of Definition 5.4 we get $k = 0$, $m = 1$, $L(G_1, T') = \neg s(x,y)$, $L(G_1, T) = \neg s(r,y)$, $L(N_a, T') = \neg p(x,v)$, and $L(N_l, T) = \neg p(r,b)$. Further, T contains no disjunctive positive literals. Selecting $\theta = \{x\backslash r, v\backslash b\}$ all conditions of Definition 5.4 are met.

On the other hand, it easy to verify that the derivation does not allow the application of forward-subsumption. Interestingly, this becomes possible if the third extension step is performed before the second one (then we get the multiset of open goals $\{\neg p(r,b), \neg q(r,y)\}$ which is subsumed by the multiset of open goals after the initialization step). Hence, the success of forward subsumption in fact depends on the employed search strategy. **(End Example)**

For an important class of computation rules, namely the local ones, it is however possible to show that each derivation belonging to the T-Context Check is also pruned by the use of forward subsumption.

Theorem 5.9 *Let $D = d_1, \ldots, d_n$ be a derivation generated by using a local computation rule and let T_i denote the tableau generated by $d_1, \ldots, d_i$, for $1 \leq i \leq n$. If D belongs to the T-Context Check then for some $1 \leq j < n$, $T_j \trianglelefteq T_n$.*

The proof of this theorem uses ideas found in [Bol *et al.*, 1991] where a similar theorem is proved in the context of SLD-resolution.

Proof. Speaking in terms of Definition 5.4, let T' be the tableau emerging from T_n by undoing the derivation steps applied to N_a and its successors. Since we assume a local computation rule, r say, T' is the tableau generated by $d_1, \ldots, d_i$ for some $1 \leq i < n$, hence $T' = T_i$. Further, we know that there is a most general substitution τ such that $L(N_a, T_i)\tau = L(N_l, T_n)$.

In the sequel, let θ_l denote the substitution used for d_l $(1 \leq l \leq n)$, let $L_G = \{L(G_1, T_i), \ldots, L(G_m, T_i)\}$ and $L_A = \{L(A_1, T_i), \ldots, L(A_k, T_i)\}$.

We first show that for each variable x that occurs in $L(N_a, T_i)$ and some element of $L_G \cup L_A$, $x\theta_{i+1} \ldots \theta_n = x\tau$. To this end, recall that the definition of the T-Context Check implies that

$$(L(N_a, T_i)\tau, L_A\theta_{i+1} \ldots \theta_n, L_G\theta_{i+1} \ldots \theta_n)$$

must be an instance of $(L(N_a, T_i), L_A, L_G)$, say $(L(N_a, T_i)\rho, L_A\rho, L_G\rho)$. Clearly

$$x\rho = \begin{cases} x\tau & \text{for } x \in var(L(N_a, T_i)) \\ x\theta_{i+1}\dots\theta_n & \text{for } x \in var(L_G \cup L_A). \end{cases}$$

Hence, for $x \in var(L(N_a, T_i)) \cap var(L_G \cup L_A)$, $x\tau = x\theta_{i+1}\dots\theta_n$. $(\star)$

Now, let S be the elements of $\{G_1, \dots, G_m\}$ which are selected before N_a by r during D and let $R = \{G_1, \dots, G_m\} - S$. Since r is local, we know that complete subrefutations of the elements of S are found before the branch containing N_a is considered. Hence, there exist some j $(i \leq j < n)$ such that the tableau T_j generated by $d_1, \dots, d_j$ has the frontier $R \cup \{N_a\}$. To prove the theorem we show that $T_j \trianglelefteq T_n$. This is achieved by proving that there is a substitution σ such that

(i) $L(N_a, T_i)\theta_{i+1}\dots\theta_j\sigma = L(N_a, T_j)\sigma = L(N_l, T_n) = L(N_a, T_i)\tau$.

(ii) for each element $G \in R \cup \{A_1, \dots, A_k\}$, $L(G, T_i)\theta_{i+1}\dots\theta_j\sigma = L(G, T_j)\sigma = L(G, T_n) = L(G, T_i)\theta_{i+1}\dots\theta_n$

Taking Definition 4.1 into account it is easy to see that in case σ exists, there is for each open branch in $T_j\sigma$ some open branch in T_n with the same open goal and the same set of disjunctive positive ancestors. Hence, we have $T_j \trianglelefteq T_n$.

Let $F = R \cup \{A_1, \dots, A_k\} \cup \{N_a\}$. Then, let $L(F_i)$ and $L(F_j)$ be the sets of literals attached to the elements of F in T_i and T_j, respectively. Further, let $L(S)$ be the set of literals attached to the elements of S and their positive disjunctive ancestors in T_i.

We define σ as follows.

$$x\sigma = \begin{cases} x & \text{if } x \notin var(L(F_j)) \\ x\theta_{j+1}\dots\theta_n & \text{if } x \in var(L(F_j)) - var(L(N_a, T_i)) \\ x\tau & \text{if } x \in var(L(F_j)) \cap var(L(N_a, T_i)) \end{cases}$$

Now we prove that using this definition of σ, (i) and (ii) are valid.

Proof of (i): Let $x \in var(L(N_a, T_i))$. Then we obviously have $x \in var(L(F_i))$. We show that $x\tau = x\theta_{i+1}\dots\theta_j\sigma$.

If $x \in var(L(F_j))$ then no substitution was applied to x during the subrefutations of the elements of S. Hence, $x\theta_{i+1}\dots\theta_j = x$ and therefore $x\theta_{i+1}\dots\theta_j\sigma = x\sigma = x\tau$. Otherwise, if $x \notin var(L(F_j))$, then $x\theta_{i+1}\dots\theta_j \neq x$, hence $x \in var(L(S))$ must hold. According to $(\star)$ we therefore know that $x\theta_{i+1}\dots\theta_n = x\tau$. Moreover, for every $y \in var(x\theta_{i+1}\dots\theta_j)$, either (a) $y \in var(L(S))$ or (b) y was in introduced during the subrefutations of the elements of S.

In case (b) we obviously have $y \notin var(L(F_i))$ and therefore $y \notin var(L(N_a, T_i))$. Then, the definition of σ gives us $y\sigma = y\theta_{j+1}\dots\theta_n$ because $y \in var(L(F_j))$:

$$y \in var(x\theta_{i+1}\dots\theta_j) \subseteq var(L(N_a, T_i)\theta_{i+1}\dots\theta_j) = var(L(N_a, T_j)) \subseteq var(L(F_j)).$$

In case (a) we either have (a1) $y \in var(L(N_a, T_i)$ or (a2) $y \notin var(L(N_a, T_i)$. In the first case (a1) we know, due to $(\star)$ (and because $y \in var(L(F_j)))$ that $y\sigma = y\tau = y\theta_{i+1}\dots\theta_n = y\theta_{j+1}\dots\theta_n$. In the second case (a2) we also get $y\sigma = y\theta_{j+1}\dots\theta_n$.

Summarizing the cases (a) and (b) we get $x\theta_{i+1}\dots\theta_j\sigma = x\theta_{i+1}\dots\theta_n = x\tau$.

Proof of (ii): Let $L(R)$ denote the set of literals attached to the nodes of $R \cup \{A_1, \ldots, A_k\}$ in T_i and let $y \in var(L(R)\theta_{i+1} \ldots \theta_j)$. We have to prove that $y\sigma = y\theta_{j+1} \ldots \theta_n$ holds. To this end, let x be some variable in $var(L(R))$ such that $y \in var(x\theta_{i+1} \ldots \theta_j)$. Now we distinguish two cases.

If $x \notin var(L(S))$, then obviously $x = x\theta_{i+1} \ldots \theta_j = y$ and therefore $y \in var(L(R))$. If $x \notin var(L(N_a, T_i)$ the definition of σ implies $y\sigma = y\theta_{j+1} \ldots \theta_n$. Otherwise we know (again, due to $(\star)$) that

$$y\sigma = y\tau = y\theta_{i+1} \ldots \theta_n = y\theta_{j+1} \ldots \theta_n.$$

If $x \in var(L(S))$, then, again, either $y \in var(L(S))$ or $y \notin var(L(S))$. In the latter case we have $y \notin var(L(N_a, T_i))$ and the definition of σ implies the desired result (note that we assumed that $y \in var(L(R)\theta_{i+1} \ldots \theta_j)$, hence $y \in var(L(F_j)))$. In the former case we again distinguish two cases: If $y \notin var(L(N_a, T_i))$ we can again apply the definition of σ. Otherwise, if $y \in var(L(N_a, T_i))$ we know that $y\theta_{i+1} \ldots \theta_j = y$ (because y appears in $var(L(R)\theta_{i+1} \ldots \theta_j))$ and due to $(\star)$ we have $y\sigma = y\tau = y\theta_{i+1} \ldots \theta_n = y\theta_{j+1} \ldots \theta_n$ which is the desired result. **Q.E.D.**

We have seen that the T-Context Check is compatible with blockwise and disjunctive regularity. In what follows we prove that the T-Context Check is even – at least partially – able to prevent derivations violating blockwise regularity on its own.

That this cannot be achieved in general is illustrated by the next example. Afterwards we show however that each derivation which violates a restricted version of blockwise regularity belongs to the T-Context Check.

Example 5.10 Consider a clause set containing the following clauses.

$$\begin{array}{llll} (1) & p(a) & \leftarrow & q(a) \\ (2) & p(a) & \leftarrow & p(x), q(b) \\ (3) & & \leftarrow & p(a) \end{array}$$

Suppose that after selecting clause (3) for an initialization step, a derivation D proceeds by applying two extension steps selecting the open goals $\neg p(a)$ and $\neg p(x)$ and the clauses (2) and (1), respectively. The corresponding tableaux are shown in Figure 8. Clearly, the rightmost tableau T violates blockwise regularity. It is however easy to verify that T does not belong to the T-Context Check. **(End Example)**

For a restricted version of blockwise regularity given in the following definition we can prove the desired result.[17]

Definition 5.11 (Weak Blockwise Regularity) A branch is *weakly blockwise regular* if its leaf node N belongs to a block $N_1, \ldots, N_k, N$ and the label of N is different to the label of N_i, for all $1 \leq i \leq k$.

A tableau is weakly blockwise regular if all of its open branches are weakly blockwise regular. **(End Definition)**

[17]This restricted form of regularity equals the identical ancestor pruning rule as it is defined in [Poole and Goebel, 1985].

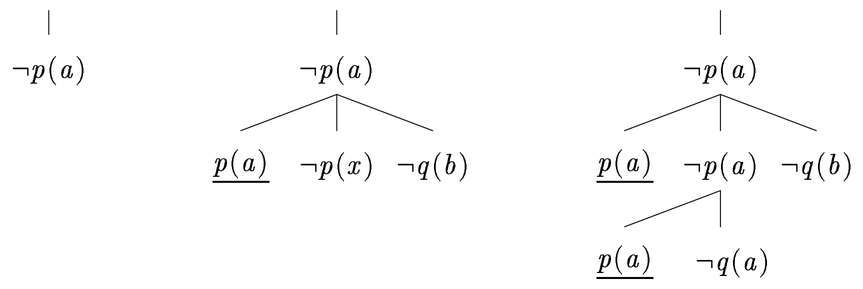


Figure 8: Tableaux for Example 5.10

Theorem 5.12 *Every derivation generating a tableau which is not weakly blockwise regular is pruned by the T-Context Check.*

Proof. Suppose a derivation D generates a tableaux T which is not weakly blockwise regular. Then, T contains some open goal L_l which is identical to one of its ancestors L_a. Further the nodes labeled with L_l and L_a, N_l and N_a say, occur in the same block. Hence, neither N_l nor N_a nor any node which is a descendant of N_a and a successor of N_l is disjunctive positive.

Now, let T' be the tableau emerging from T by removing the derivation steps applied to N_a and its descendants, let $S' = \{G_1, \dots, G_m, N_a\}$ be the frontier of T', and let σ' be the composition of substitutions needed to generate T'. Obviously, σ' is more general than the substitution σ which is the composition of substitutions needed to build T. Hence there is a substitution θ such that $\sigma'\theta = \sigma$.

Then we know that $L(N_a, T')\theta = L(N_l, T)$, $L(G_i, T')\theta = L(G_i, T)$ for all $1 \leq i \leq m$, and $L(A_i, T')\theta = L(A_i, T)$ for all $1 \leq i \leq k$, where $A_1, \dots, A_k$ are the disjunctive positive nodes occurring on open branches in T'. But this means that D belongs to the T-Context Check. **Q.E.D.**

With Theorem 5.9 we could now conclude that each tableau which was generated using a local computation rule and violates weak blockwise regularity can be deleted by the use of forward subsumption. But one can do better. In the following theorem we show that in case local computation rules are considered, each tableau that violates *full* blockwise regularity is deleted by subsumption.

Theorem 5.13 *Let T be a tableau generated by a MES derivation S which uses a local computation rule. If T is not blockwise regular then there exists a fair derivation such that T is subsumed by some tableaux in S_∞.*

Proof. Suppose a tableau T generated by a subsumption ME derivation violates blockwise regularity. Then there exist two nodes, N_1 and N_2 say, in one block of T such that N_1 is an ancestor of N_2 and $L(N_1, T) = L(N_2, T)$. If N_2 is a leaf node of T we immediately can conclude the desired result by applying Theorem 5.9 and Theorem 5.12 since we assume a local computation rule.

Otherwise, let $d_1, \ldots, d_n$ be the sequence of derivation steps applied to N_2 and its successors in order to generate T. Since N_1 and N_2 belong to the same block, and $L(N_1, T) = L(N_2, T)$ it is obviously possible to apply a sequence of (unrestricted) derivation steps $d'_1, \ldots, d'_n$ directly to N_1 where d'_i equals d_i except that d'_i uses the substitution σ which is the composition of all substitutions used to generate T. Let T' denote the resulting tableau. Since a local computation rule is assumed (derivation steps can only be applied to N_1 and its successors until each branch containing N_1 is closed), it is easy to see that $T' \trianglelefteq T$ must hold.

Then, there also exists a tableau T'' which is more general than T' and which is generated by applying a derivation $d''_1, \ldots, d''_n$ to N_1 where d''_i equals d'_i but uses a MGU (note that the application of such a derivation is possible since we assume local computation rules to be stable under substitution). Clearly, $T''\theta \trianglelefteq T$ holds for some substitution θ.

Unfortunately, it is not clear that T'' is contained in S_∞ because its generation might be pruned. We can show however that in this case there exists another tableau in S_∞ which subsumes T. Let T''_i be the tableau generated after d''_i for $1 \leq i \leq n$. Suppose T''_i is subsumed by some other tableau $T_i \in S_\infty$. If $i = n$, T_i immediately also subsumes T and we are done.

If $i < n$ we distinguish two cases. Let OB_i be some subset of $OB(T''_i) - b$ where b is the branch to which d''_{i+1} is applied. If $OB(T_i\theta) \geq OB_i$ for some substitution θ (case 1), then T_i also subsumes T and we are done. Otherwise (case 2), it is possible to apply a derivation step d to T which corresponds to the application of d''_{i+1} to T''_i. The resulting tableau T_{i+1} obviously subsumes the tableau T''_{i+1}. If T_{i+1} is not persistent, it is pruned by some other tableau T_x. But this tableau T_x also subsumes T''_{i+1}. Hence in all cases we either know that T is subsumed by some persistent tableau or that T''_{i+1} is subsumed by some persistent tableau. In the latter case we can iterate our argumentation for $i+1, i+2, \ldots$ until we found a persistent tableau which subsumes T''_n (note that $T''_n = T$). **Q.E.D.**

Note, however, that this theorem does not imply that an additional check which discards non-regular tableaux would be useless. The fairness condition of MES only guarantees that the subsuming tableau is generated at *some* point during a derivation. Hence it might not be possible to *subsume* T immediately after its generation (and therefore further tableaux might be built from T). Instead it could be removed immediately by an additional check for blockwise regularity.

5.1 Reachability Analysis

The disjunctive positive refinement in fact considerably reduces the amount of literals which are needed for reduction steps during ME deductions. However, there are situations where even disjunctive positive ancestor literals can be ignored. Such a situation is illustrated in the following example.

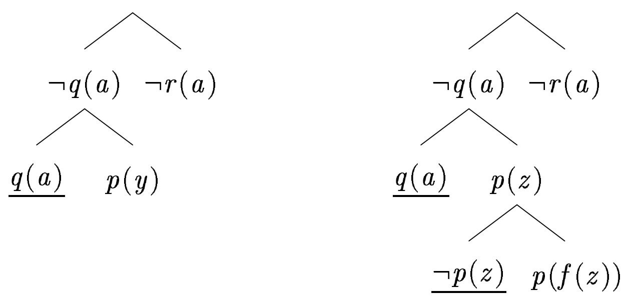


Figure 9: Tableaux for Example 5.14

Example 5.14

$$\begin{array}{lrcl}
(1) & r(a) & \leftarrow & \\
(2) & q(a) & \leftarrow & p(f^5(b)) \\
(3) & p(f(z)) & \leftarrow & p(z) \\
(4) & q(a), p(y) & \leftarrow & \\
(5) & & \leftarrow & q(x), r(x)
\end{array}$$

The task is to find a refutation of the above clause set with top-clause (5). Applying an initialization step and an extension step which uses clause (4) results in the tableau depicted in Figure 9 on the left hand side. A further extension step applied to the open goal $p(y)$ gives the tableau T which is shown on the right. Unfortunately we now cannot apply the same arguments as in Example 5.1 in order to prune the derivation. This is because $p(z)$ is disjunctive positive. However, it is easy to verify that no open goal generated during a subrefutation for $p(f(z))$ can be made complementary to $p(z)$. Taking the disjunctive positive refinement into account, this means that none of the ancestors in T is relevant for the application of reduction steps; hence the derivation can be pruned. **(End Example)**

In order to identify the ancestor literals which might be used by a reduction step we use the concept of *reachability* originally proposed in [Neugebauer, 1992].[18] Intuitively speaking, a literal L is reachable from a literal K if L may occur as subgoal of K.

Definition 5.15 (Reachability, Relevance) Let K be a literal and $\mathcal{S}$ a clause set. The set of literals which are *reachable* from K (in $\mathcal{S}$) is defined inductively.

- K is reachable.
- If a literal L is reachable and there exists a literal L' contained in a clause $C \in \mathcal{S}$ such that L^d and L' are weakly unifiable, then all literals in $C - \{L'\}$ are reachable, too.
- No other literals are reachable.

[18]Similar concepts have been studied in [Bibel and Buchberger, 1985] and [Sutcliffe, 1992].

We call a node N in a tableau T *relevant*, if the label of N can be made complementary to a literal which is reachable from some open goal in T. **(End Definition)**

Clearly, if for some open goal G, there is no literal reachable from G which can be made complementary to a disjunctive positive ancestor G' of G, then (an instance of) G' definitely cannot be used for a reduction step during a subrefutation for G. Thus, reachability is a useful means to check which ancestor literals may become relevant to find a subrefutation for an open goal. In a similar form it has been successfully used in [Sutcliffe, 1992] to detect goals whose subrefutations can be found by exclusively applying Horn clause techniques.

Example 5.16 We continue Example 5.14. The literals reachable from the literal $p(f(z))$ of clause (3) are the literals $q(a)$ in clause (2), $\neg r(x)$, and the literal $p(f(z))$ itself. Hence, we learn that during a subrefutation of $p(f(z))$ (in the tableau depicted in Figure 9 on the right hand side) no open goal can be made complementary to (an instance of) $p(z)$. Therefore, no ancestor in the tableau is needed for the application of reduction steps and the derivation can be pruned. **(End Example)**

Summarizing, we can define a variant of the T-Context Check, called *T*-Context Check*, whose definition equals Definition 5.4 except that the term "disjunctive positive" is replaced by "disjunctive positive and relevant".[19]

Corollary 5.17 *The completeness of blockwise regular ME is preserved even if each derivation belonging to the T*-Context Check is pruned.*

Note 5.18 Note, however, that we did not claim that the completeness of blockwise *and* disjunctive regular ME is preserved by the T*-Context Check (this is only of interest if disjunctive regular ME is complete). This is only the case if we restrict disjunctive regularity as follows: After an application of the T*-Context Check, the irrelevant disjunctive positive literals contained in the considered branch must not be used to check disjunctive regularity. **(End Note)**

Note 5.19 Given a clause set $\mathcal{S}$, most of the work needed to examine whether some literal L is reachable from another literal K can be shifted into a preprocessing step. One has to build up a table containing one row and one column for each literal in the clause set. An entry of this table is set to 1 if the literal corresponding to the column is reachable from the literal corresponding to the row, otherwise to 0.

At runtime, given an instance $K\sigma$ of a literal K in $\mathcal{S}$, there are two possibilities to check whether another literal L is reachable from $K\sigma$. The first (and less precise) possibility is to ignore the substitution σ and to look up the table entry for K and L. If one does not want to neglect σ, one has to check (by inspecting the table) whether L is reachable from a literal K', where K' occurs in a clause C which contains a further literal L' which is weakly unifiable with $K^d\sigma$. **(End Note)**

[19]The T*-Context Check is a proper generalization of a loop check proposed in [Brüning, 1993]. The loop check presented there augments the Context Check of [Besnard, 1989] by a reachability analysis but does not make use of the positive refinement of ME.

6 Experimental Results

Model Elimination with Subsumption. The ME calculus with forward and backward subsumption (MES) of Section 4 is implemented in our "The_Mission" prover (*The*orem proving by *M*odel El*i*mination with *S*ub*s*umpt*ion*). The_Mission features general theory handling as described in [Baumgartner, 1995], lemmas, factorization and a built-in weak regularity check. As discussed previously, the subsumption mechanisms covers blockwise regularity. Nevertheless we have also a built-in weak regularity check in order to relate the pruning power of subsumption to the more easily implementable (and widely used) special case of weak regularity. In order to explore the power of subsumption proper, we employed neither lemmas nor factorization in the examples below.

The_Mission was designed as an experimental device for examining the use of subsumption in connection with model elimination. Hence the main requirement was flexibility, not performance. Therefore it is written as a Meta-Interpreter in Prolog. Due to the need for a sound unification we used ECRC's Eclipse-Prolog. In order to speed up retrieval of candidates for subsumption checks, we employed term indexing [Graf, 1994]. Nevertheless, our impression is that the implementation could be speeded up significantly by using an implementation language which supports *destructive* data manipulation better than Prolog. Hence the runtimes given below should not be read as the ultimate results for ME with subsumption.

One interesting feature of the Meta-interpreter approach is that it allows to guide the search better than usual PTTP-approaches. More specifically, we employ a weighting function which selects among all tableaux in store the lightest one, and applies all possible inferences to some selected branch in that tableau. A fairly good (and fair!) strategy is to select a minimal tableau wrt. the sum of "depth" (i.e. the length of a longest branch) and "width" (i.e. length of the front). This function was used throughout all our experiments.

In the construction of the tableaux sets we trace for each tableau the history of its generation. That is, if a new tableau is stored, we keep the information from which tableau it was generated and by what kind of inference. Thus, for each tableau we have a list of its predecessors which lead to its generation. This list is useful not only for printing a trace at the end, but also supports a heuristic for forward subsumption: it turned out (empirically) that without loosing too many successful subsumption checks it suffices to restrict the check whether a tableau T is subsumed by some already stored tableau to the predecessors of T. This is plausible, because tableaux not being in this predecessor relation are often developed to tableaux which are not similar enough for a successful subsumption check. A notable exception is the example in Table 1 labeled with "$^-$".

Table 1 summarizes the results. The examples run are almost the same as for the T-Context Check prover above. The examples run were all taken from the TPTP library [Sutcliffe *et al.*, 1994].

Column 1 contains the problem name. If marked with a star ("*") we have chosen a clause different from the clause labeled as "theorem" in the TPTP-library, because otherwise a proof wouldn't exist. Column 2 and 3 contain the results for plain disjunctive positive ME without and refinements (but reduction steps are allowed to

	DPME							*SETHEO*	
	Plain		*Block. Reg.*		*withSubsumption*				
Problem	*Inf.*	*Time*	*Inf.*	*Time*	*Inf.*	*Subs.*	*Time*	*Inf.*	*Time*
NUM020-1*	58	< 1	42	1	42	$8 + 0$	< 1	52	< 1
NUM021-1*	$\perp$		2163	5.6	1870	$308 + 15$	10	574	< 1
NUM002-1	1639	4.5	748	2.5	711	$72 + 0$	4.5	817	< 1
RNG010-1*	124	< 1	94	< 1	94	$13 + 0$	< 1	3097	< 1
GRP031-2	671	1.9	547	1.5	695	$25 + 6$	5.8	6290	< 1
PLA001-1	$\perp$		$\perp$		1092	$193 + 0$	11.7	39474	1
PLA003-1	970	2.8	362	1.2	66	$30 + 0$	< 1	391	< 1
MSC001-1$^-$	$\perp$		$\perp$		860	$214 + 8$	18	108481	10
MSC001-1					3254	$668 + 47$	29		
MSC002-1	1565	4.9	1565	4.9	135	$3 + 0$	< 1	495	< 1
MSC006-1	845	2.3	537	1.5	1263	$90 + 24$	11	4526	< 1

*: other clause than the given 'theorem' clause used as top clause.
$^-$: forward subsumption among ALL tableaux generated so far.
$\perp$: abort due to memory overflow.
Inf.: Number of inferences (extension + reduction steps) to find proof.
Time: Runtime on a Sparc 10-20 in seconds.
Subs.: $M + N$ means M (N) forward (backward) subsumptions.

Table 1: Table of runtime results for various versions of the disjunctive positive refinement of ME (DPME) in combination with subsumption deletion.

non-disjunctive nodes as well). In column 4 and 5 weak blockwise regularity was used. Column 6, 7 and 8 list the results for forward and backward subsumption ME with the heuristics and tableaux selection function as described above. Note again that The_Mission enumerates tableaux, not derivations. Hence we found it interesting to relate our approach to SETHEO [Letz *et al.*, 1992], a high inference rate, compiling ME prover which enumerates derivations (column 9 and 10). We used version 3.2. in standard flag setting. This setting includes clause set preprocessing (reordering) and employs regularity, subsumption and tautology constraints. We switched off "folding up" (a generalized form of factorization) since it did not improve the results significantly. Furthermore, the results wrt. inference steps are better related to our prover, where we also switched off factorization.

Let us comment on the runtimes in Table 1. When comparing DPME with DPME with weak regularity it is confirmed once more that the (weak) regularity check considerable cuts the number of inferences to find a proof. Since it is cheap to implement, runtimes improve roughly to the same degree. On the other hand, the forward and backward subsumption proper are not that cheap to implement. It is thus not surprising that runtimes can easily get longer. There seems to be a turnover-point (cf. MSC002-1) where the benefits of saved tableau expansions due to subsumption outweigh the additional costs. However, we are optimistic to improve on the runtime results by a more clever implementation.

Let us now turn to the inference counts. The result for DPME with subsumption show that in almost all cases the number of inferences to find a proof decreases considerable. Hence, the regularity check does *not* cover the subsumption deletion to a large degree. In the best case (MSC002-1, "Blind hand problem"), subsumption deletion helped to cancel the inference number to less than a tenth. In other cases, we had to abort due to memory overflow when not using subsumption. However, there are examples where subsumption deletion is counterproductive (MSC006-1, and others not mentioned in Table 1). In order to explain this phenomenon we have to point out that for the subsumption test only disjunctive positive ancestor literals are considered, whereas reduction steps are allowed to *any* literal along a branch[20] (cf. Note 3.3). Hence, it might happen what a tableau containing important ancestor context is prevented from further expansion due to subsumption.

Finally, the last two columns contain the data for SETHEO. Since SETHEO enumerates derivation by iterative deepening (wrt. tableau depth), and all but the very last tried derivation is without success, a huge number of inferences (extension plus reduction steps) results. In the extreme case (MSC001-1$^-$) about 150 as many inferences were necessary.

To conclude, we think that subsumption is a promising direction to improve model elimination based theorem proving.

T-Context Check. The T-Context Check has been integrated into a prototypical implementation of DPME. This proof system additionally allows for checking (full) blockwise regularity in order to relate the pruning power of the T-Context Check to the one of blockwise regularity. Since the main purpose of our investigations was to explore the usefulness of the T-Context Check rather than to built a high-performance system, we used PROLOG as programming language. A further simplification resulted from the fact that we restricted our proof system to local computation rules. This allowed us (as mentioned in note 5.3) to implement the T-Context Check in a straightforward way.

Table 2 summarizes the results. All of them were computed on a SUN SPARC-station 20. The first column contains the names of the used problems. They include almost all of which were used for the experiments with The_Mission; the additional ones were also taken from the TPTP library [Sutcliffe *et al.*, 1994]. The second column indicates the applied search strategy. We found it interesting to employ two different (widely used) search strategies: "d" stands for the usage of an iterative deepening search strategy iterating over the depth of the generated tableaux whereas "l" means that the search strategy iterates over the length of the derivation. The third column contains the maximal tableaux-depth/derivation-length explored by our prover. An additional "*" indicates that a proof was found at the given limit. The results generated without using the T-Context Check are given in the fourth and fifth column. The sixth and seventh column show the corresponding numbers produced by applying the T-Context Check additionally. Finally, the last column of the table

[20]Optional reduction steps help to find a refutation more quickly in almost all examples.

lists the number of derivations which where pruned by the T-Context Check. During our experiments, we always used only the given 'theorem' clauses (that is the clauses which are marked as 'theorem' in the TPTP library) as top clauses, but no other negative clauses.[21]

To comment our results, there are some examples which obviously benefit from the use of the T-Context Check. For instance, this is the case for PLA001-1 or MSC002-1 were proofs can be found with considerably less inference steps *and* time (for comparison see also the number of inference steps used by SETHEO in table 1). In other cases, where a proof could not be found (within the given inference/depth limits) the number of inference steps is reduced to a large extent, e.g. see NUM020-1 or LCL038-1. These results demonstrate that blockwise regularity is in many cases not able to cover the T-Context Check. Clearly, there are also examples where the T-Context Check is not very useful; however, the run-time results show that the additional computational overhead was neglectable in all cases.

Summarizing, these experiments demonstrate, that a weak variant of subsumption deletion (namely the T-Context Check) can be useful for reducing both inference steps *and* proof time. A comparison of table 1 and table 2 however shows that the pruning power of subsumption deletion as used in The_Mission is in fact stronger than the one of the T-Context Check (e.g. see PLA001-1 or MSC002-1). Therefore it might not only be interesting to improve the implementation of The_Mission to reduce the computational requirements for (full) subsumption deletion. Additionally it might be worthwhile to think about something "between" the T-Context Check and subsumption deletion, that is an extension of the T-Context Check which retains its simplicity (in view of an efficient implementation) and covers more of the cases were subsumption deletion can be applied successfully.

7 Discussion

In this paper we successfully adapted the notion of subsumption known from resolution based approaches to Model Elimination. It soon became clear that an approach comparing entire Model Elimination tableaux by subsumption would be useless. On the other hand, completeness is lost if only the open goals of tableaux without their ancestors, i.e. the frontiers are taken into account.

In order to overcome these problems we first introduced a variant of Model Elimination which minimizes ancestor literals. In [Plaisted, 1990], D. Plaisted showed that it suffices to keep only *positive* literals as ancestors. This obviously allows to define a larger subsumption relation because only the positive ancestors contained in tableaux have to be obeyed for the subsumption tests. We further improved on this by restricting the ancestor context even more, namely to those positive literals which stem from disjunctive clauses. Importantly, this allows to ignore all positive ancestors which stem from definite clauses. The resulting calculus, the so-called disjunctive positive refinement of model elimination was proved to be sound and complete.

[21]Note that this implies that the results concerning NUM020-1, NUM021-1, and RNG010-1 in table 1 and table 2 cannot be compared. In case another negative clause is used as top-clause, the proofs for these theorems can be found fast (0.1, 6.1, and 0.5 seconds, respectively) with the proof system containing the T-Context Check.

			Block. Reg. only		plus T-Context Check		
Problem	M.	Limit	Inf.	Time	Inf.	Time	Pruned
NUM020-1	l	9	176489	205.1	96219	111.1	4418
	d	4	9382	6.0	2447	1.6	45
NUM021-1	l	8	47096	77.9	31780	51.6	1409
	d	4	10787	8.7	2883	2.3	29
NUM002-1	l	7*	16897	19.0	16897	19.6	0
	d	3*	649	0.6	649	0.6	0
RNG010-1	l	6	23958	42.7	22978	41.2	220
	d	4	35853	63.6	35853	64.4	0
GRP031-2	l	7*	1871	1.6	1871	1.8	0
	d	3*	2716	1.9	2716	2.0	0
GRP040-3	l	5	10866	25.1	9917	23.2	27
	d	3	248168	159.9	246759	158.7	101
PLA001-1	l	12*	>500000	>600	39803	89.9	9409
	d	7*	6031	7.2	1709	2.3	130
PLA003-1	l	7*	343	0.8	127	0.3	18
	d	7*	658	1.0	197	0.3	18
COL005-1	l	8	79406	75.1	61491	59.8	13
	d	4	248168	156.1	246759	158.1	101
LCL105-1	l	14	72257	143.1	69435	140.1	639
	d	5	76845	165.9	66700	145.0	810
LCL038-1	l	20	131020	254.7	115180	245.8	2106
	d	7	>250000	>600	86551	107.7	3598
MSC001-1	l	10	12873	38.9	7406	22.6	42
	d	6*	4911	5.7	2469	3.0	21
MSC002-1	l	11*	5771	12.7	1625	3.8	20
	d	6*	777	0.9	244	0.3	6
SET012-2	l	7	22764	77.6	22716	77.7	62
	d	5	28200	21.5	28200	21.6	0
CAT001-1	l	7	103720	278.0	103514	283.7	25
	d	4	715823	745.5	715823	754.0	0
CAT015-3	l	5	23556	54.5	17644	39.5	366
	d	4	>500000	>600	345555	205.7	605

Table 2: Table of results for an implementation of DPME in combination with the T-Context Check.

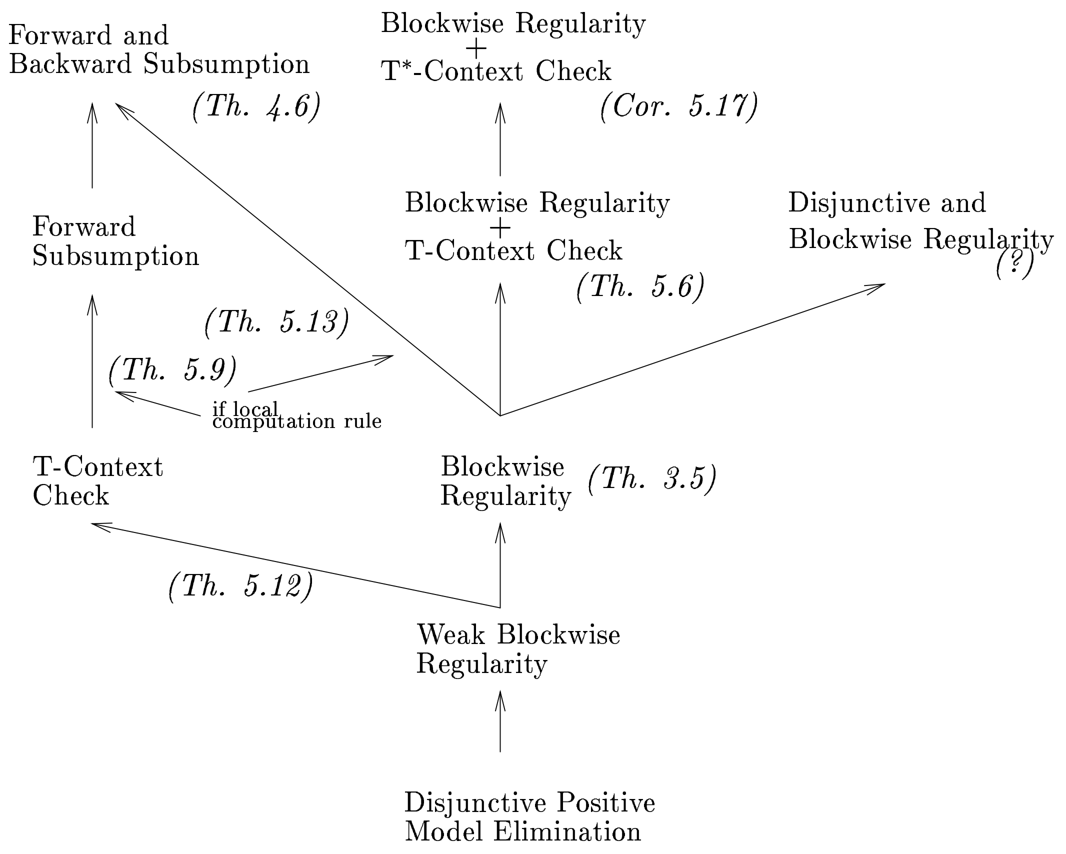
With this refinement of Model Elimination at hand, we defined a calculus which integrates forward and backward subsumption. This required a new organization of the proof process: Instead of constructing *one* tableau by nondeterministically guessing the inference steps and backtracking on failure, all tableaux that can be built are stored *explicitly*. By this we changed Model Elimination from an enumeration procedure into a saturation procedure which allows to delete all generated tableaux that are subsumed by some other generated tableau. It was proven that using a fair selection strategy, this new calculus, called subsumption Model Elimination, is sound and complete.

As an alternative to subsumption Model Elimination we proposed a variant of forward subsumption which, instead of comparing entire frontiers (together with their disjunctive positive ancestors), aims at pruning a derivation if a subrefutation for only one open goal would constitute a subrefutation for one of its ancestors. We proved that the use of the resulting pruning technique, the T-Context Check, does not affect completeness of disjunctive positive Model Elimination. Further, we showed that the T-Context Check is an instance of forward subsumption (as used in subsumption Model Elimination) in case local computation rules are employed. We also related a very important refinement, namely regularity, to the Context Check and the use of subsumption. We were able to prove that weak blockwise regularity is an instance of the T-Context Check and that each tableau which violates blockwise regularity is deleted by the use of subsumption. Finally, we proposed a refinement of the T-Context Check which allows to ignore even some of the positive disjunctive ancestors in a tableau by applying a reachability analysis.

The relations between the investigated refinements of Model Elimination are depicted in Figure 10. There are three top elements, namely the forward and backward subsumption, the blockwise regularity and T*-Context check, and the disjunctive and blockwise regular refinement. According to the semantics of the "$\longrightarrow$" notation, these are the most restrictive, and hence most attractive, variants. The completeness of the disjunctive and blockwise regular refinement is still open. The remaining two variants are incomparable wrt. pruning power, but they both are complete. These are the main results of this paper. These calculi are radically different, because the one enumerates tableaux and the other enumerates derivations. Hence we will not prefer the one to the other and we conclude that both of them are interesting.

To illustrate the potential of the proposed refinements we presented comprehensive experimental results which confirm that in many cases search spaces are considerably reduced. In particular, our experiment show that subsumption deletion in many cases allows to concentrate on only few tableaux which can be built with only few inference steps. Further, the run-times given in table 2 show, that weaker variants of subsumption deletion can be implemented very easily without resulting in high computational overheads.

Subsumption deletion, however, not only aims at reducing search spaces to find proofs more easily. Additionally, it is sometimes quite useful to detect that a theorem *cannot* be proven. In fact, it has been shown in [Bol *et al.*, 1991] that (a restricted variant of) the T-Context Check prunes every infinite Model Elimination derivation of various natural classes of clause sets without function symbols. These classes include Horn-clause sets that do not introduce new variables during a derivation (the so-called



Forward and Backward Subsumption
(Th. 4.6)
Blockwise Regularity + T*-Context Check
(Cor. 5.17)
Forward Subsumption
Blockwise Regularity + T-Context Check
(Th. 5.6)
Disjunctive and Blockwise Regularity
(?)
(Th. 5.13)
(Th. 5.9)
if local computation rule
T-Context Check
Blockwise Regularity
(Th. 3.5)
(Th. 5.12)
Weak Blockwise Regularity
Disjunctive Positive Model Elimination

Figure 10: Relating the “pruning power” of the various refinements. A relationship “$A \rightarrow B$” means: whenever a derivation is pruned by the refinement A then it is also pruned by the refinement B. Either this relation holds trivially, and in this case the link remains undecorated, or else the respective result from the text is cited. For trivial links a respective completeness result is stated at the more restrictive refinements, which entail completeness of the refinements below them.

nvi clause sets), and Horn-clause sets that allow at most one recursive call per clause (the so-called *recursion-restricted Horn-clause sets* which, for example, include all Horn-clause sets of which all rules have a body with at most one literal)[22]. These results are relevant for the non-Horn case, too, since it happens quite frequently that a subproblem occurring during the deduction of a non-Horn clause set can be proved using extension steps only because of its structure and the use of the disjunctive positive refinement (and therefore this subproblem “behaves” like a Horn clause set).

Due to theorem 5.9 such results can be immediately applied to subsumption Model Elimination in case local computation rules are used. Further, it might be possible to

[22]To achieve the completeness result one has to assume a selection function which selects the recursive literal at last.

prove much stronger results for subsumption Model Elimination since (as depicted in Figure 10) the pruning power of forward and backward subsumption is stronger than the one of the T-Context Check. This can be nicely illustrated by the second clause set ("equality") given in Example 4.5. As we have shown there, the satisfiability of this clause set can be decided by subsumption Model Elimination. This is not the case for ordinary Model Elimination augmented by the T-Context Check. The T-Context Check only allows to (roughly) halve the number of applied inference steps but is not able to prune every infinite derivation.

8 Future Work

Equality. As always, much remains to be done. In Example 4.5 we identified, as an instance of subsumption deletion, situations where extension steps with the cumbersome equality axioms are unnecessary. A respective restriction of paramodulation which forbids paramodulation into and below variables was suggested. This has to be investigated more rigorously.

Extended Subsumption. It is possible to extend the backward subsumption rule. Suppose in a fair derivation T subsumes T'. Backward subsumption can be extended to delete not only T, *but also all tableaux which are generated from* T*, except for* T'. In order to be complete, forward subsumption has to be restricted to predecessing tableaux. This calculus has to be defined formally.

Instantiations A problem which we have not tackled in this paper is the ability to take into account instantiations of open goals which are caused by further derivation steps. Even if some tableau T does not subsume another tableau T', this might be the case if substitutions are considered which are applied to T' by additional extension or reduction steps. In fact, it has been illustrated in [Brüning, 1994] that taking such additional instantiations into account can result in a considerable strengthening of the pruning power.

Example 8.1 Consider a clause set containing the following clauses.

$$\begin{array}{lccl} (1) & q(a) & \leftarrow & r(b) \\ (2) & q(c) & \leftarrow & p(a), q(z) \\ (3) & p(y) & \leftarrow & q(c) \\ (4) & & \leftarrow & p(x), q(x) \end{array}$$

Suppose that, after selecting clause (4) for an initialization step, a derivation D proceeds by applying three extension steps selecting the open goals $\neg p(x)$, $\neg q(c)$ and $\neg q(z)$ and the clauses (3), (2), and (1), respectively. The generated sequence of multisets of open goals is

$$\begin{array}{ll} (a) & \{\neg p(x), \neg q(x)\}, \\ (b) & \{\neg q(c), \neg q(y)\}, \\ (c) & \{\neg p(a), \neg q(z), \neg q(y)\}, \\ (d) & \{\neg p(a), \neg r(b), \neg q(y)\}. \end{array}$$

Since none of these multisets subsumes another one, subsumption is not applicable (furthermore, the tableau generated by this derivation is not simply subsumed by another tableau which can be constructed by applying different extension steps). However, the variable y occurring in multiset (c) is instantiated to a by the last extension step. This instantiated multiset is subsumed by multiset (a). Thus, subrefutations for the goals in (c) which use this last extension step can be directly applied to the elements of (a). Hence, D is redundant. **(End Example)**

Interestingly, it can be shown that a refined version of the T-Context Check, which allows for considering additional substitutions, is able to prune every derivation that violates *full* blockwise regularity. Recall that this is not the case for the T-Context Check as defined in the section 5.

Answer Computing In [Baumgartner *et al.*, 1995] the restart model elimination calculus is used for computing answers. It is demonstrated there that computing an answer for a given problem can be much harder than finding only a refutation (i.e. proving that an answer *exists*). It would be interesting to investigate the calculi presented here wrt. answer computing.

Acknowledgments

Thanks to M. Kühn and B. Thomas who implemented the Mission Prover. Further we thank A. de Waal for carefully reading a previous version of this paper.

References

[Antoniou and Langetepe, 1994] G. Antoniou and E. Langetepe. Applying SLD-Resolution to a Class of Non-Horn Logic Programs. *Bulletin of the IGPL*, 2(2):231–243, 1994.

[Astrachan and Stickel, 1992] O. L. Astrachan and M. E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In D. Kapur, editor, *Proceedings of the Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 224–238. Springer, 1992.

[Bachmair, 1991] L. Bachmair. *Canonical Equational Proofs.* Progress in Theoretical Computer Science. Birkhäuser, 1991.

[Baumgartner and Furbach, 1993] P. Baumgartner and U. Furbach. Consolution as a Framework for Comparing Calculi. *Journal of Symbolic Computation*, 16(5), 1993. Academic Press.

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

[Baumgartner and Furbach, 1994b] P. Baumgartner and U. Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *LNAI*, pages 769–773. Springer, 1994.

[Baumgartner *et al.*, 1995] P. Baumgartner, U. Furbach, and F. Stolzenburg. Model Elimination, Logic Programming and Computing Answers. In *Proceedings of IJ-CAI '95*, 1995. (to appear, Long version in: Research Report 1/95, University of Koblenz, Germany).

[Baumgartner, 1994] P. Baumgartner. Refinements of Theory Model Elimination and a Variant without Contrapositives. In A.G. Cohn, editor, *11th European Conference on Artificial Intelligence, ECAI 94*. Wiley, 1994. (Long version in: Research Report 8/93, University of Koblenz, Institute for Computer Science, Koblenz, Germany).

[Baumgartner, 1995] P. Baumgartner. Linear and Unit-Resulting Refutations for Horn Theories. *Journal of Automated Reasoning*, 1995 (?). (To appear, also in: Research Report 9/93, Institute for Computer Science, University of Koblenz, Germany).

[Besnard, 1989] P. Besnard. On infinite loops in logic programming. Technical Report 488, IRISA, Rennes, France, 1989.

[Bibel and Buchberger, 1985] W. Bibel and B. Buchberger. Towards a connection machine for logical inference. *Future Generations Computer Systems Journal*, 1(3):177–188, 1985.

[Bol *et al.*, 1991] R. N. Bol, K. R. Apt, and J. W. Klop. An analysis of loop checking mechanisms for logic programming. *Journal of Theoretical Computer Science*, 86:35–79, 1991.

[Brüning, 1993] S. Brüning. On Loop Detection in Connection Calculi. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proceedings of the Kurt Gödel Colloquium*, pages 144–151. Springer Verlag, 1993.

[Brüning, 1994] S. Brüning. *Techniques for Avoiding Redundancy in Theorem Proving Based on the Connection Method.* PhD thesis, TH Darmstadt, 1994.

[Chang and Lee, 1973] C. L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Fronhöfer and Caferra, 1988] B. Fronhöfer and R. Caferra. Memorization of literals: An enhancement of the connection method. Technical report, Institut für Informatik, Technische Universität München, 1988.

[Fronhöfer, 1985] B. Fronhöfer. On refinements of the connection method. In J. Demetrovics, G. Katona, and A. Salomaa, editors, *Algebra, Combinatorics and Logic in Computer Science*, pages 391–401, Amsterdam, 1985. North-Holland.

[Furbach *et al.*, 1989] Ulrich Furbach, Steffen Hölldobler, and Joachim Schreiber. Horn equational theories and paramodulation. *Journal of Automated Reasoning*, 3:309–337, 1989.

[Gallier, 1987] J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Wiley, 1987.

[Graf, 1994] P. Graf. Extended Path-Indexing. In *Automated Deduction – CADE 12*, volume 814 of *Lecture Notes in Artifical Intelligence.* Springer, 1994.

[Letz *et al.*, 1992] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2), 1992.

[Letz *et al.*, 1994] R. Letz, K. Mayr, and Ch. Goller. Controlled Integrations of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13(3):297–338, 1994. Special Issue on Automated Reasoning with Analytic Tableaux.

[Letz, 1993] R. Letz. *First-Order Calculi and Proof Procedures for Automated Deduction.* PhD thesis, TH Darmstadt, 1993.

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

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

[Loveland, 1986] D. W. Loveland. Mechanical theorem proving by model elimination. *Journal of the ACM*, 15:236–251, 1986.

[Mayr, 1995] K. Mayr. Link Deletion in Model Elimination. In R. Hähnle P. Baumgartner and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, volume 918 of *Lecture Notes in Artificial Intelligence*, pages 169–184, 1995.

[Neugebauer, 1992] G. Neugebauer. From Horn Clauses to First Order Logic: A Graceful Ascent. Technical Report AIDA–92–21, FG Intellektik, FB Informatik, TH Darmstadt, 1992.

[Overbeek and Wos, 1989] R. Overbeek and L. Wos. Subsumption, a Sometimes Undervalued Procedure. Technical Report MCS-P93-0789, Argonne National Laboratory, Argonne, IL., 1989.

[Plaisted, 1990] D. Plaisted. A Sequent-Style Model Elimination Strategy and a Positive Refinement. *Journal of Automated Reasoning*, 4(6):389–402, 1990.

[Poole and Goebel, 1985] D. Poole and R. Goebel. On eliminating loops in PROLOG. *Sigplan Notices*, 20(8):38–40, 1985.

[Robinson and Wos, 1969] G. A. Robinson and L. Wos. Paramodulation and Theorem Proving in First Order Theories with Equality. In Meltzer and Mitchie, editors, *Machine Intelligence 4.* Edinburg University Press, 1969.

[Stickel, 1988] M. E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, Ch. Suttner, and T. Yemenis. The TPTP problem library. In A. Bundy, editor, *Proceedings of the Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 252–266. Springer Verlag, 1994.

[Sutcliffe, 1992] G. Sutcliffe. Linear-Input Subset Analysis. In D. Kapur, editor, *Proceedings of the Conference on Automated Deduction*, pages 268–280. Springer, 1992.

[van de Riet, 1993] R. P. van de Riet. An overview and appraisal of the fifth generation computer system project. *Future Generation Computer Systems Journal*, 9:83–103, 1993.