Controlled Integrations of the Cut Rule into Connection Tableau Calculi

R. LETZ, K. MAYR, and C. GOLLER
Institut für Informatik, Technische Universität München, Germany
e-mail: letz@informatik.tu-muenchen.de

Abstract. In this paper techniques are developed and compared which increase the inferential power of tableau systems for classical first-order logic. The mechanisms are formulated in the framework of connection tableaux, which is an amalgamation of the connection method and the tableau calculus, and a generalization of model elimination. Since connection tableau calculi are among the weakest proof systems with respect to proof compactness, and the (backward) cut rule is not suitable for the first-order case, we study alternative methods for shortening proofs. The techniques we investigate are the folding up and the folding down operation. Folding up represents an efficient way of supporting the basic calculus, which is top-down oriented, with lemmata derived in a bottom-up manner. It is shown that both techniques can also be viewed as controlled integrations of the cut rule. In order to remedy the additional redundancy imported into tableau proof procedures by the new inference rules, we develop and apply an extension of the regularity condition on tableaux and the mechanism of anti-lemmata which realizes a subsumption concept on tableaux. Using the framework of the theorem prover SETHEO, we have implemented three new proof procedures which overcome the deductive weakness of cut-free tableau systems. Experimental results demonstrate the superiority of the systems with folding up over the cut-free variant and the one with folding down.

Key words. Theorem proving, first-order logic, tableaux, connection method, model elimination, lemmata, cut rule.

1 Introduction

Tableau and connection calculi offer interesting approaches to automated deduction which deviate from the uniform resolution paradigm. A convincing illustration of this fact is the relative success of proof procedures based on *model elimination* [Loveland, 1968, Loveland, 1978], as described in [Stickel, 1988], [Letz et al., 1992], or [Astrachan and Loveland, 1991]. Although, historically, introduced against the background of resolution as a procedure *generating formulae*, in its essence model elimination belongs to the family of *analytic* tableau and connection calculi. The most elegant format for introducing a generalized form of model elimination is by using the framework of *connection tableaux*, which we elaborate in this paper. The framework results from integrating concepts of the connection method [Bibel, 1981, Bibel, 1987] into the tableau calculus [Beth, 1955, Beth, 1959], [Smullyan, 1968]. Connection tableau procedures are successful and promising because of their goal-orientedness and the existence of powerful techniques for reducing the search space.

Due to their *cut-freeness*, however, connection tableau calculi are significantly weaker concerning deductive power than resolution systems, so that for many problems proofs cannot be found simply because no short proofs exist in the calculus. The (backward) cut rule, which could remedy this weakness, is not suitable for first-order logic, since it cannot be controlled in a reasonable way. Therefore, alternative methods of shortening proofs

have to be found. In this paper, we develop and compare two controlled variants of the cut rule, the *folding up operation* and the *folding down operation*, which can be applied without introducing too much additional indeterminism. The folding up operation is presented as an efficient way of *integrating lemmata* into the connection tableau calculus. Folding up generalizes the *C-reduction* operation introduced in [Shostak, 1976] for the model elimination format. Folding down can be viewed as an alternative formulation of the *factorization rule* used in connection calculi and model elimination while producing an additional reductive effect. We show that folding up (and hence C-reduction) has properly more deductive power than factorization and folding down.

While additional inference rules may guarantee the existence of shorter proofs, it is typically at the price of increasing the search space. Consequently, we investigate whether the gain in proof compactness obtained by our additional inference rules is really beneficial when the problem of *finding* proofs is addressed. For this purpose different ways of eliminating redundancy in connection tableau procedures have to be discussed. In this paper we concentrate on two pruning techniques. The first one is directly motivated by the additional inference rules, since the manner in which folding up and folding down are applied gives rise to a new structural refinement of tableaux, the condition of *strong regularity*, which can be used for pruning the proof *search*. The other technique results from an adaptation of the concept of *subsumption* to tableau procedures. Subsumption is a general mechanism of high significance for any form of proof search, but particularly important if additional redundancy is imported, as is the case for additional inference rules. Since full subsumption seems to difficult to be integrated efficiently into tableau procedures, we develop a refined variant of subsumption, the mechanism of so-called *anti-lemmata*. Using anti-lemmata one can avoid repeated solutions of subgoals with the same or more specialized instantiations in the search process. However, there are compatibility problems of subsumption and anti-lemmata with different forms of regularity, which we uncover. These results naturally lead to the development of three new connection tableau procedures, which we have implemented on top of the model elimination theorem prover SETHEO [Letz et al., 1992]. Experiments with a number of representative examples from the field of automated deduction demonstrate that the two systems with folding up perform significantly better than the cut-free variant of connection tableaux and the one with folding down. Thus the new procedures can easily solve problems which are out of scope or at least extremely difficult for pure top-down oriented cut-free tableau procedures, like the intermediate value theorem or the algebraic problems wos31 and wos33.

The paper is organized as follows. The connection tableau framework and some of its refinements, like the connection calculus and model elimination, are introduced in the second section. Then we present variants of the cut rule, as important reference points for the techniques developed later on. In Section 4, we formulate factorization in the connection tableau framework, and demonstrate that factorization is a restricted form of the cut rule. The folding up operation is introduced in the fifth section, as an efficient way of integrating bottom-up lemmata into the top-town oriented methodology of connection tableaux; furthermore, we prove that the folding up operation is superior to factorization with respect to inferential power, but still a refined variant of the atomic cut rule. In Section 6, by a slight modification of the folding up technique the folding down operation is obtained. In Section 7, the notion of strong regularity is introduced, as an interesting

new search pruning mechanism, which is compatible with the folding up operation, for depth-first selection functions; the completeness proof of the new calculus is given, which is very short and hence illustrates the elegance of the connection tableau framework; we also describe a possible combination of folding up and folding down. In the eighth section the subsumption concept is adapted to the tableau framework, and the compatibility problems of subsumption with the structural tableau refinements are discussed; furthermore, the mechanism of anti-lemmata is introduced, as a efficient approximation of subsumption. Section 9 contains a brief description of the implementation of the new mechanisms. Section 10 is devoted to the presentation of experimental results and their assessment. We conclude with a summary of the presented results and some remarks on future promising inference and reduction methods for tableau procedures.

2 The Connection Tableau Framework

Since our interest is in proof systems for sets of *first-order clauses*, we have to fix the meaning of some basic notions.

A *literal* is an atomic formula A or the negation $\neg A$ of an atomic formula. Given a literal L, then the *complement* $\sim L$ of L is the negation of L if L is an atom, and the atom of L otherwise. *Clauses* are defined inductively as follows. Every literal is a *clause*. If L is a literal and c is a clause, then $L \vee c$ is a *clause*; we do not consider the empty clause. A literal L is said to be *(contained) in* a clause if L occurs as a disjunct in the clause. The fundamental representation format we are using is that of a tree labelled with literals.

Definition 2.1 (Literal tree) A *literal tree* is a pair $\langle t, \lambda \rangle$ consisting of an ordered (downward) tree t and a labelling function λ assigning 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 predecessor N, in the order given by t. The *depth* of a node N in a (literal) tree is the number of nodes dominating N.

2.1 Tableaux as Declarative Proof Objects

In order to simplify the presentation and the completeness proofs, we distinguish between tableaux as static *proof objects* and tableau *calculi* which consist of inference rules for constructing tableaux. Also, in contrast to the standard way of generalizing the tableau calculus from the ground case to the first-order case, by including different rules for quantifier elimination [Smullyan, 1968], the working with Skolemized formulae renders the first-order calculus significantly simpler and also facilitates the incorporation of *unification* into the tableau calculus. First, we introduce tableaux as declarative objects.

Definition 2.2 ((Clausal) tableau) A *(clausal) tableau T for* a set of clauses S is a literal tree $\langle t, \lambda \rangle$ in which, for any successor sequence of nodes $N_1, \ldots, N_n$ in t labelled with literals $K_1, \ldots, K_n$, respectively, there is a substitution σ and a clause $L_1 \vee \cdots \vee L_n \in S$ with $K_i = L_i\sigma$, for every $1 \leq i \leq n$. $K_1 \vee \cdots \vee K_n$ is called a *tableau clause*. A tableau is said to be *closed* if on every branch a literal and its complement appear.

Tableaux are *refutational complete* for sets of clauses, i.e., for any unsatisfiable set of clauses there is a closed tableau for the set. Completeness can be preserved for a number of *structural refinements* of tableaux (see [Letz, 1993a]). The most important for search pruning are the following two.

Definition 2.3 (Connection tableau) A tableau T is called *connected* or a *connection tableau* if each inner node N labelled with a literal L has a leaf node N' among its successor nodes which is labelled with the literal $\sim L$.

Definition 2.4 (Regular tableau) A tableau is *regular* if no two nodes on a branch are labelled with the same formula.

In Figure 1 a closed regular connection tableau for a set of clauses is depicted.

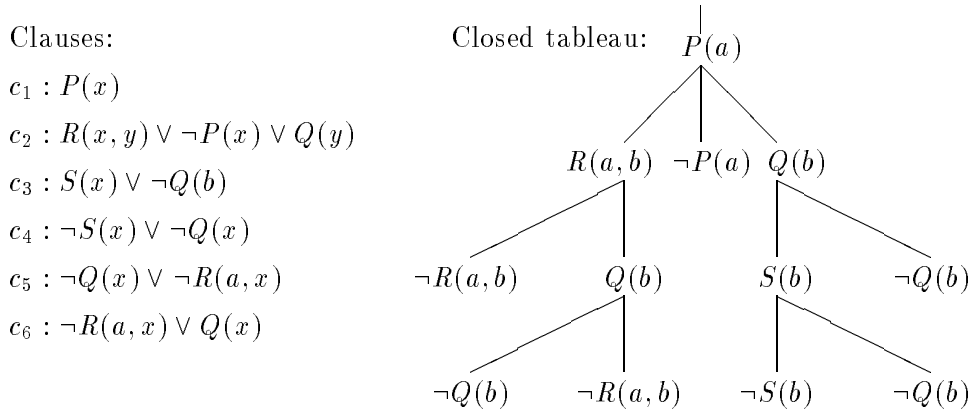

Figure 1: Closed regular connection tableau for a set of clauses.

While the regularity restriction does not decrease the *deductive power* of general tableaux, i.e., every smallest closed tableau for a set of clauses is regular, it is important to note that the *connectedness* condition may have such weakening effects. In order to express this formally, we have to refer to the standard abstract notion for assessing the relative deductive power of proof systems.

Definition 2.5 (Polynomial simulation) A proof (refutation) system $\mathcal{P}_1$ *polynomially simulates* a proof (refutation) system $\mathcal{P}_2$ if there is a polynomial p such that for any valid (inconsistent) formula F with a proof (refutation) D_1 in $\mathcal{P}_1$ there is a proof (refutation) D_2 of (an appropriate translation of)[1] F in $\mathcal{P}_2$ with $\text{size}(D_2) < p\,(\text{size}(D_1))$.

In [Letz, 1993a], the following two fundamental results are proven.

1. Connection tableaux cannot polynomially simulate tableaux.

2. Regular connection tableaux cannot polynomially simulate connection tableaux.

This illustrates the complementarity of improving deductive and reductive power.

[1]Translations may be necessary if the systems work on different languages or if one of the systems proves formulae directly and the other refutes formulae. The *appropriateness* of translations is investigated in [Reckhow, 1976], it is not relevant here since we are uniformly dealing with sets of clauses.

2.2 Tableau Calculi

The static specifications of tableaux and connection tableaux put no particular restrictions on the instantiations that may be applied to the renamed clauses from the input set in their use as tableau clauses. The procedural counterparts of the static deduction objects, however, can be defined using *unification* as instantiation operation, this way achieving finite branching rates of the calculi. The two inference rules of the tableau *calculus (with unification)* are tableau *expansion* and tableau *reduction*. For the formulation of the calculus, we introduce the notion of *marked* tableaux.

Definition 2.6 (Marked tableau) A *marked tableau* is a pair $\langle T, \mu \rangle$ consisting of a tableau T and a partial labelling function μ from the set of leaf nodes of T into the set of nodes of T. A (branch with) leaf node N of a marked tableau is called *marked* if $N \in \text{domain}(\mu)$. The unmarked leaf nodes in a marked tableau are called *subgoals*. A node N in a marked tableau is said to be *solved* if either N is marked or all leaf nodes dominated by N; the marked (sub)tableau rooted in a solved node N is called a *(sub)proof* of N. A marked (sub)tableau is named *solved* if its root node is solved.

With the marking of a leaf node in a tableau we indicate that the respective branch has indeed been checked off as being inconsistent. The markings also are fundamental for the *extensions* of the tableau calculi considered later on.

Procedure 2.1 (Tableau expansion) Given a set S of clauses as input and a marked tableau T for S, select a subgoal N, choose a clause $c \in S$, obtain a variant $L_1 \vee \cdots \vee L_n$ of c in which the variables are disjoint from the variables in the literals occurring in T, then attach n new successor nodes $N_1, \ldots, N_n$ to N and label them with $L_1, \ldots, L_n$ respectively.

Procedure 2.2 (Tableau reduction) Given a marked tableau T, select a subgoal N with literal L, choose a dominating node N' with literal L' such that there is a most general unifier σ for L and $\sim L'$, then apply the substitution σ to the tableau literals and mark N with N'.

The *connection tableau calculus* consists of three inference rules, the tableau reduction rule plus the following two inference rules, which result from restrictions and combinations of the two given tableau construction rules, respectively.

Procedure 2.3 (Tableau start) Given a set of clauses S as input and a one-node tree with a root only, simply perform a tableau expansion step.

Procedure 2.4 (Tableau extension) Given a set of clauses S as input and a marked connection tableau T for S, select a subgoal N with literal L, apply a tableau expansion step at N immediately followed by a tableau reduction step at one of the new nodes N' with its predecessor N, and mark N' with N.

Apparently, the (connection) tableau calculus can only generate marked (connection) tableaux.

2.3 Subgoal Selection Functions

There is a source of indeterminism in the (connection) tableau calculus which can be removed without any harm concerning proof length. This indeterminism concerns the selection of the next subgoal at which an expansion, extension, or reduction step is to be performed.

Definition 2.7 (Subgoal selection function) A *(subgoal) selection function* ϕ is a mapping assigning a subgoal to every marked tableau T which is not solved.

Although with the two (three) inference rules of the (connection) tableau calculus not *every* (connection) tableau can be generated—this is due to the working with most general unifiers—, the inference rules are adequate with respect to the static specifications of (connection) tableaux, in the following manner.

Definition 2.8 A literal tree T is *more general* (*strictly more general*) than a literal tree T' if the unordered trees (ordered trees) underlying T and T' are isomorphic and there is a variable substitution σ such that for any pair of associated nodes N in T and N' in T' with literals L and L', respectively: $L\sigma = L'$.

Proposition 2.1 *Given any marked (connection) tableau T for a set of clauses, then, for any subgoal selection function, there exists a sequence of inference steps in the (connection) tableau calculus and an output tableau T' which is strictly more general than T.*

A proof of this relation between the static tableau proof objects and their operational counterparts can be found, e.g., in [Letz, 1993a].

Note As a corollary of this proposition, we obtain a strong form of independence of the subgoal selection function which is extremely important for tableau proof procedures. Thus, we can work with *any* selection function without losing deductive power.[2]

Definition 2.9 (Standard selection functions) A *depth-first (depth-last) selection function* selects from any marked tableau T containing subgoals one with a maximal (minimal) depth. The *depth-first (depth-last) left-most (right-most) selection function* selects the left-most (right-most) subgoal from the ones with maximal (minimal) depth.

2.4 Subgoal Trees and Subgoal Formulae

For proving the unsatisfiability of a set of formulae using the tableau framework, it is not necessary to *explicitly construct* a closed tableau, it is sufficient to *know* that the deduction *corresponds* to a closed tableau.

Definition 2.10 (Subgoal tree) The *subgoal tree of* a marked tableau T is the literal tree obtained from T by deleting out all solved nodes, together with their ingoing edges.

[2] This strong independence does not hold for resolution calculi, like, e.g., linear resolution, which is only *weakly* independent of the selection function, in the sense that all selection functions (of literals to be resolved upon) do preserve completeness, but they cannot produce the same proof objects. For linear resolution, it is not even guaranteed that one selection function polynomially simulates the other ones.

The subgoal tree of a tableau contains only the subgoals of a tableau and those nodes which dominate subgoals. Since for the continuation of the refutation process, all other parts of the tableau may be disregarded without any harm[3], most implementations of tableau calculi work by manipulating subgoal trees instead of tableaux.

From subgoal trees, however, there is but a small step to interpret them as logical formulae. There exists a natural injective mapping from subgoal formulae to logical formulae over the connectives $\neg$, $\vee$ and $\wedge$, so-called *subgoal formulae*, inductively defined as follows.

Definition 2.11 The *subgoal formula* of the empty subgoal tree is $\bot$, the *subgoal formula* of a one-node tree is the label of its node (if existing), and the *subgoal formula* of any complex subgoal tree S is the formula $(L \wedge D)$ where L is the label of the root of S (if existing) and D is the disjunction of the subgoal formulae of the immediate subgoal subtrees of N.

Note An alternative formula-oriented recasting of subgoal trees is to transform them into so-called *consolvents* [Eder, 1991] which are essentially the disjunctions of the conjunctions of the literals on the branches of the subgoal trees. The consolvent reformulation, however, is not injective, as opposed to the subgoal formula representation which exactly mirrors the subgoal tree structure.

2.5 Tableaux and Related Formalisms

The connection tableau format is closely related with two other well-known frameworks in automated deduction, namely, *connection matrices* and *model elimination chains*.

2.5.1 Connection Matrices

The *connection calculus* presented in [Bibel, 1987] Chapter III.6 can be viewed as a version of the connection tableau calculus restricted to depth-first selection functions. Here we shall consider a refinement of this connection calculus, without *factorization*, which is studied below. The favourite notation for displaying connection proofs is by writing them as *matrices*, with the columns consisting of the literals in the clauses. The information about the paths which have been examined and those which remain to be checked in a certain state, is expressed with some additional data structures. In the original presentation, some worked-off parts remain noted in the deduction. We apply the same technique used for tableaux—the working with subgoal trees—to the connection calculus, by working with *subgoal matrices*. The resulting format is particularly appealing for the presentation of deductions obeying any form of depth-first selection, as shown in Figure 2. In the matrix proof we have indicated the *next* inference with arrows at the selected subgoal, $\rightarrow\!\triangleright$ for extension, $\triangleleft\!\leftarrow \cdots —$ for reduction, and $=\!\circ\!=$ for the path under consideration, the so-called *active* path. Additionally, in any extension step, the clause and the entry literal is given, and in any reduction step the respective predecessor node. The relation of subgoal matrix proofs with the more general subgoal tree notation is evident: a subgoal

[3] *Completely* ignoring solved parts of a tableau, however, may increase the search space, since certain cases of irregularity might be overlooked (cf. Subsection 8.2).

matrix is basically a subgoal tree put on its left side. The relation with subgoal formulae is apparent, too. To give an example, the subgoal formula which corresponds to the fourth subgoal matrix in Figure 2, is the formula $P(a) \wedge (Q(y) \vee (R(a,y) \wedge Q(y) \wedge \neg R(a,y)))$.

$P(x)$	$\multimap$	$c_2.2$				
$P(x)$	$=\!\!\circ$	$Q(y)$				
		$R(x,y)$	$\multimap$	$c_6.1$		
$P(a)$	$=\!\!\circ$	$Q(y)$				
		$R(a,y)$	$=\!\!\circ$	$Q(y)$	$\multimap$	$c_5.1$
$P(a)$		$Q(y)$				
		$R(a,y)$	$\circ\!\!=$	$Q(y)$	$=\!\!\circ$	$\neg R(a,y)$
$P(a)$	$=\!\!\circ$	$Q(y)$	$\multimap$	$c_3.2$		
$P(a)$	$=\!\!\circ$	$Q(b)$	$=\!\!\circ$	$S(x')$	$\multimap$	$c_4.2$
$P(a)$	$=\!\!\circ$	$Q(b)$	$\circ\!\!=$	$S(x')$	$=\!\!\circ$	$\neg Q(x')$
$\bot$						

Figure 2: Subgoal matrix notation of a depth-first construction of the tableau in Figure 1.

2.5.2 Model Elimination Chains

The *model elimination calculus* was introduced in [Loveland, 1968] and improved in [Loveland, 1969, Loveland, 1978] and [Astrachan and Loveland, 1991]. Model elimination can be viewed as a refinement of the connection tableau calculus in which subgoals are selected in a *depth-first right-most* manner. This re-interpretation of model elimination has various advantages concerning generality, elegance, and the possibility for defining extensions and refinements.[4] Although the original model elimination format works with a special "chain" notation in which square brackets are used instead of binary logical connectives and punctuation symbols, model elimination essentially manipulates subgoal formulae. The close relation with subgoal formulae is evident from the following simple transformation.

Transformation from model elimination chains to subgoal formulae
While working through a model elimination chain from right to left, iteratively apply the following operation.

> Replace every substring $L_1 \cdots L_n$ with a disjunction $(L_1 \vee \cdots \vee L_n)$, and every substring $[L]F$ with a conjunction $(L \wedge F)$.

The transformation also works in the other direction.

Note Since the subgoal selection function determines the branching rate of the search tree of the calculus, i.e., the number of possible tableaux that can be generated in one inference step from a given tableau (see Definition 8.2), all forms of restrictions of the

[4]The soundness and completeness results for model elimination, for example, are immediate consequences of the soundness and completeness proofs of connection tableaux, which are very short and simple, as opposed to the rather involved and lengthy proofs in [Loveland, 1978].

subgoal selection from the side of the *framework* may heavily reduce the potential of search pruning in the calculi of this framework. It is important to emphasize that due to its greater freedom of choosing between selection functions, the (connection) tableau (or subgoal formulae) format is superior to frameworks supporting depth-first selection functions only, like connection matrices and model elimination chains. In [Letz, 1993a] it is demonstrated that certain formulae have exponential search trees for *any* depth-first selection function, whereas certain simple *free* selection functions exist which induce only search trees of a linear size.

3 The Cut Rule and the Connection Cut Rule

The regular connection tableau calculus has proven successful in the practice of automated deduction [Stickel, 1988, Letz et al., 1992], although concerning deductive power the calculus is extremely weak. In fact, even general tableaux cannot polynomially simulate the method of *truth tables*. One of the simplest classes that demonstrates this fact is given in Example 3.1.

Example 3.1 For any set $\{A_1, \ldots, A_n\}$ of distinct propositional atoms, let S_n denote the set of all 2^n clauses of the shape $L_1 \vee \ldots \vee L_n$ where $L_i = A_i$ or $L_i = \neg A_i$, $1 \leq i \leq n$.

The elements of this class have small (i.e. linear) truth tables but only exponential closed tableaux, as illustrated in Figure 3 (for a proof in more detail see, e.g., [Letz, 1993a]).

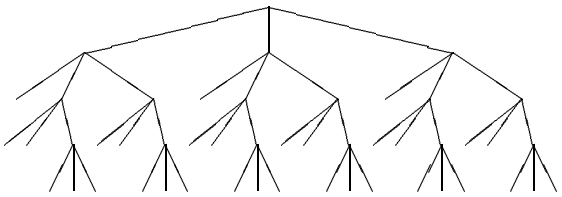

Figure 3: Tree structure of a minimal closed tableau for Example 3.1, $n = 3$.

Tableaux can be made significantly stronger with respect to deductive power by adding the backward variant of the *cut rule* from the sequent system [Gentzen, 1935]. Since we consider *clausal* tableaux, only the *atomic* variant of the cut rule is relevant here.[5]

Procedure 3.1 ((Atomic) cut) Given a marked tableau T, select a subgoal N, choose an arbitrary (atomic) formula A, attach two successor nodes, and label them with A and $\neg A$, respectively. A is named the *cut formula* of the cut step.

It is apparent that the connectedness condition on tableaux blocks any reasonable application of the atomic cut rule. This is because the connectedness condition enforces that any application of cut at a subgoal N labelled with a literal L must label the two

[5] In [Letz, 1993b] it is shown that in propositional logic general tableaux with full cut can polynomially simulate the sequent system with cut and *lemmata*, i.e., where deductions are not required to be *trees* of sequents but *directed acyclic graphs* of sequents.

attached successor nodes with the literals L and $\sim L$, respectively, with the result that absolutely no advance is made towards the solving of the tableau. The effect of the cut rule on the shortening of tableau proofs can only be achieved for connection tableaux if the cut rule is introduced in a form compatible with the connectedness condition.

Procedure 3.2 ((Atomic) connection cut) Given a marked tableau T, select a subgoal N with literal L, choose an arbitrary (atomic) formula A, and perform an extension step with the formula $\sim L \vee A \vee \neg A$. A is named the *cut formula* of the connection cut step.

Proposition 3.1 *(Regular) connection tableaux with the (atomic) connection cut rule can linearly simulate tableaux with (atomic) cut (and hence semantic trees).*

It is clear that the connection cut is an inference rule of a theoretical value only, since the uncontrolled addition of the connection cut to the connection tableau rules completely destroys the good reductive properties of the calculus. The question is whether there exist other forms of additional inference rules which are better suited for automated deduction.

4 Factorization

The *factorization* rule was used in model elimination [Loveland, 1978] and in the connection calculus [Bibel, 1987], Chapter III.6, but, due to the restrictions of the formats, only for depth-first selection functions. On the general level of the (connection) tableau calculus, which permits arbitrary node selection functions, the rule can be motivated as follows. Consider a closed tableau containing two nodes N_1 and N_2 with the same literals as labels. Furthermore, suppose that all ancestor nodes of N_2 are also ancestors of N_1. Then, the closed tableau part T below N_2 could have been reused as a solution and attached to N_1, because all expansion and reduction steps performed in T under N_2 are possible in T under N_1, too. This observation leads to the introduction of *factorization* as an additional inference rule. Factorization permits to mark a subgoal N_1 as solved if its literal can be unified with the literal of another node N_2, provided that the set of ancestors of N_2 is a subset of the set of ancestors of N_1; additionally, the respective substitution has to be applied to the tableaux. Used on the set of clauses

$$\{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q\}$$

which denotes an instance of Example 3.1, for $n = 2$, factorization yields a shorter proof, as shown in Figure 4. Factorization steps are indicated with arcs. Obviously, in order to preserve soundness the rule must be constrained to prohibit solution cycles. Thus, in Figure 4 factorization of the subgoal N_4 on the right hand side with the node N_3 with the same literal on the left hand side is not permitted after the first factorization (node N_1 with node N_2) has been performed, because this would involve a reciprocal, and hence unsound, employment of one solution within the other. To avoid the cyclic application of factorization, tableaux have to be supplied with an additional factorization dependency relation.

Definition 4.1 (Factorization dependency relation) A *factorization dependency relation on a tableau* T is a strict partial ordering $\prec$ on the tableau nodes.

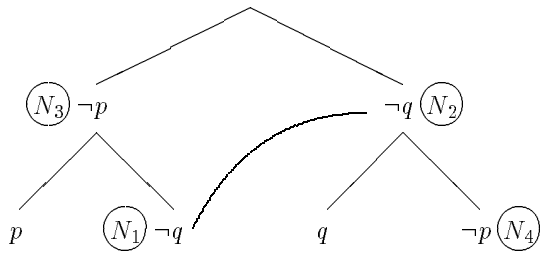

Figure 4: Factorization step in a connection tableau for Example 3.1, $n = 2$.

Procedure 4.1 (Tableau factorization) Given a marked tableau T and a factorization dependency relation $\prec$ on its nodes. First, select a subgoal N_1 with literal L and another node N_2 labelled with a literal K such that

1. there is a most general unifier σ: $L\sigma = K\sigma$,
2. N_1 is dominated by a node N which has the node N_2 among its immediate successors, and
3. $N_3 \not\prec N_2$, where N_3 is the brother node of N_2 on the branch from the root to N_1, including the latter.[6]

Then, mark N_1 with N_2 and modify $\prec$ by first adding the pair of nodes $\langle N_2, N_3 \rangle$ and then forming the transitive closure of the relation; finally, apply the substitution σ to the tableau. We say that the subgoal N_1 has been *factorized with* the node N_2. The tableau construction is started with an empty factorization dependency relation, and all other tableau inference rules leave the factorization dependency relation unchanged.

Applied to the example shown in Figure 4, when the subgoal N_1 is factorized with the node N_2, the pair $\langle N_2, N_3 \rangle$ is added to the previously empty relation $\prec$, thus denoting that the solution of the node N_3 depends on the solution of the node N_2. After that, factorization of the subgoal N_4 with the node N_3 is not possible any more.

Note It is clear that the factorization dependency relation only relates brother nodes, i.e., nodes which have the same immediate predecessor.

The applications of factorization at a subgoal N_1 with a node N_2 can be subdivided into two cases. Either, the node N_2 has been solved already, or the node N_2 or some of the nodes dominated by N_2 are not yet marked. In the second case we shall speak of an *optimistic* application of factorization, since the node N_1 is marked as solved *before* it is known whether a solution exists. Conversely, the first case will be called a *pessimistic* application of factorization.

Similar to the case of ordinary (connection) tableaux, if the factorization rule is added, the order in which the tableau rules are applied does not influence the structure of the tableau.

Proposition 4.1 (Strong node selection independency of factorization) *Any closed (connection) tableau with factorization for a set of clauses constructed with one selection function can be constructed with any other selection function.*

[6] Note that N_3 may be N_1 itself.

Switching from one selection function to another may mean that certain optimistic factorization steps become pessimistic factorization steps and vice versa.

Note If we are working with subgoal trees, i.e., completely remove solved parts of a tableau, as done in the chain format of model elimination, then for all *depth-first* selection functions solely optimistic applications of factorization can occur. Also, the factorization dependency relation may be safely ignored, because the depth-first procedure and the removal of solved subgoals render cyclic factorization attempts impossible. It is for this reason, that the integration approaches of factorization into model elimination or the connection calculus have not mentioned the need for a factorization dependency relation. Note also that if factorization is integrated into the chain format of model elimination, then the mentioned strong node selection independency does not hold, since pessimistic factorization steps cannot be performed.

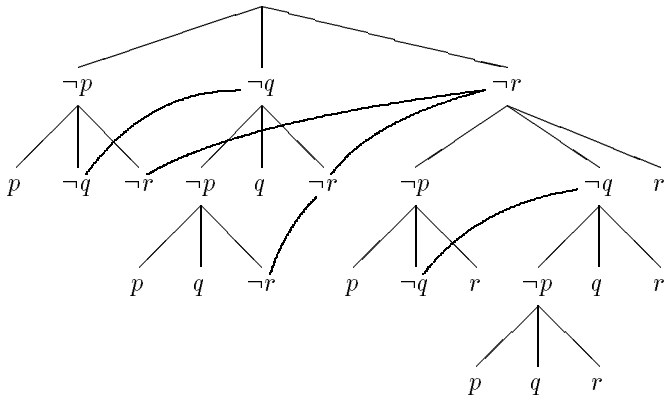

Figure 5: Linear closed connection tableau with factorization for Example 3.1, $n = 3$.

The addition of the factorization rule increases the deductive power of (connection) tableaux significantly. Thus, the critical class for tableaux given in Example 3.1, for which no polynomial proof exists (see Figure 3), has linear closed connection tableaux with factorization, as shown in Figure 5. With the factorization rule connection tableaux linearly simulate truth tables. In fact, the factorization rule can be viewed as a certain restricted version of the cut rule.

Proposition 4.2 *Tableaux with atomic cut (and regular connection tableaux with atomic connection cut) linearly simulate (connection) tableaux with factorization.*

Proof We perform the simulation proof for tableaux with cut. Given a closed tableau with factorization, each factorization step of a node N_1 with a node N_2, both labelled with a literal L, can be simulated as follows. First, perform a cut step with $\sim L$ and L at the ancestor N of N_2, producing new nodes N_4 and N_5; thereupon, move the tableau part formerly dominated by N below N_4; then, remove the tableau part underneath N_2 and attach it to N_5; finally, perform reduction steps at N_1 and N_2. The simulation is graphically shown in Figure 6. □

As shown in [Letz, 1993c], tableaux with factorization can even polynomially simulate tableau with atomic cut. *Connection* tableaux with factorization, however, cannot polynomially simulate tableaux with atomic cut, as will be proven in the next section.

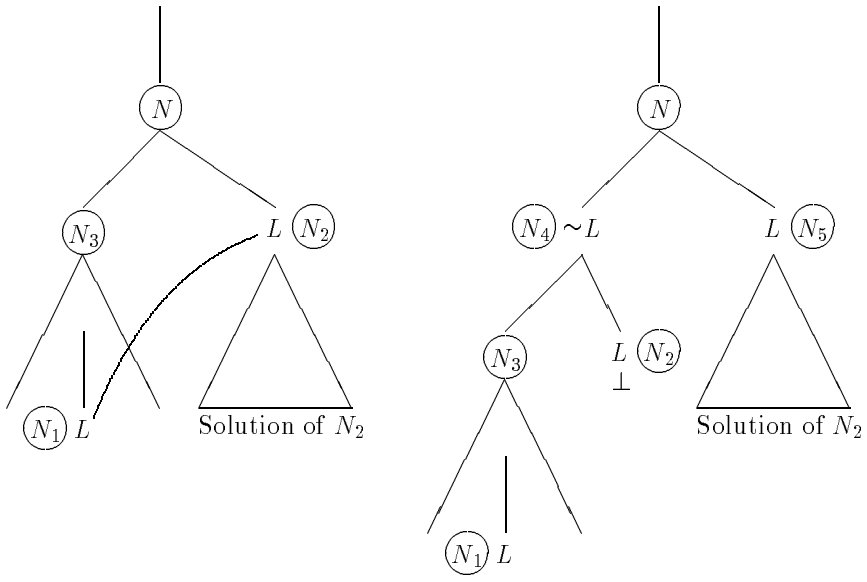

Figure 6: Simulation of factorization by cut.

5 The Folding Up Rule

An inference rule which, for connection tableaux, is stronger than factorization concerning deductive power, is the so-called *folding up rule* (in German: "Hochklappen"). Folding up generalizes the *C-reduction* rule introduced to the model elimination format in [Shostak, 1976]. In contrast to factorization, for which pessimistic and optimistic application do not differ concerning deductive power, the shortening of proofs achievable with folding up results from its pessimistic nature. The theoretical basis of the rule is the possibility of extracting *bottom-up lemmata* from solved parts of a tableau, which can be used on other parts of the tableau (as described in [Loveland, 1978] and [Letz et al., 1992], or [Astrachan and Loveland, 1991]). Folding up represents a particularly efficient realization of this idea.

We explain the rule with an example. Given the tableau displayed on the left of Figure 7, where the bold arrow points to the node at which the last inference step (a reduction step with the node 3 levels above) has been performed. With this step we have solved the dominating subgoals labelled with the literals r and q. In the solutions of those subgoals the predecessor labelled with p has been used for a reduction step. Apparently, this amounts to the derivation of two lemmata $\neg r \vee \neg p$ and $\neg q \vee \neg p$ from the underlying formula. The new lemma $\neg q \vee \neg p$ could be added to the underlying set and subsequently used for extension steps (this has already been described in [Letz et al., 1992]). The disadvantage of such an approach is that the new lemmata may be *non-unit* clauses, as in the example, so that extension steps into them would produce new subgoals, together with an unknown additional search space. The redundancy brought in this way can hardly be controlled.

With the folding up rule a different approach is pursued. Instead of adding lemmata of arbitrary lengths, so-called *context unit lemmata* are stored. In the discussed example, we may obtain two context unit lemmata:

$\neg r$, valid in the *(path)* context p, and

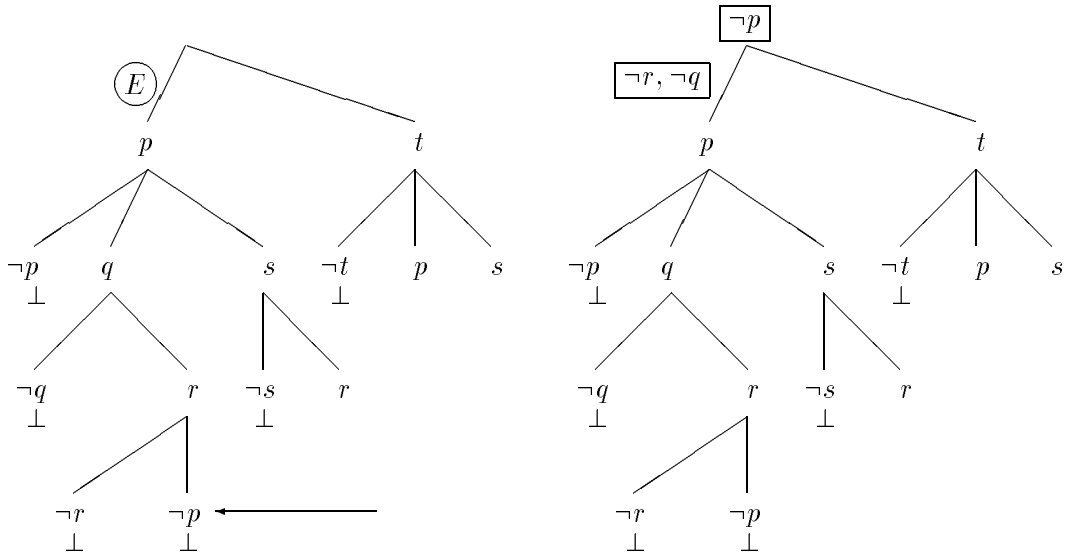

Figure 7: Connection tableau before and after three times folding up.

$\neg q$, valid in the context p.

Also, the memorization of the lemmata is not done by augmenting the input formula but *within* the tableau itself, namely, by "folding up" a solved subgoal to the edge which dominates its solution context. More precisely, the folding up of a solved subgoal N to an edge E means labelling E with the negation of the literal at N. Thus, in the example above the edge E above the p-node on the left-hand side of the tableau is successively labelled with the literals $\neg r$ and $\neg q$, as displayed on the right-hand side of Figure 7; lists of context-unit lemmata are depicted as framed boxes. Subsequently, the literals in the boxes at the edges can be used for ordinary reduction steps. So, at the leaf node labelled with r a reduction step can be performed with the edge E, which was not possible before the folding up. After that, the subgoal s could also be folded up to the edge E, which we have not done in the figure, since after marking that subgoal the part below E is completely solved. But now the p-subgoal on the left is solved, and we can fold it up above the root of the tableau; since there is no edge above the root, we simply fold up *into* the root. This folding up step facilitates that the p-subgoal on the right can be solved by a reduction step.

The gist of the folding up rule is that only *unit* lemmata are added, so that the additionally imported indeterminism is not too large. Over and above that, the technique gives rise to a new form of pruning mechanism called *strong regularity*, which is discussed below. Lastly, the folding up operation can be implemented very efficiently, since no renaming of literals is performed, as in a full lemma mechanism.

In order to be able to introduce the inference rule formally, we have to slightly generalize the notion of tableaux.

Definition 5.1 ((Marked) edge-labelled tableau) A *(marked) edge-labelled tableau (E-tableau)* is just a (marked) tableau as introduced in the Definitions 2.2 and 2.6, with the only modifications that also the edges and the root node are labelled by the labelling function λ, namely, with lists of literals.

Procedure 5.1 (E-tableau folding up) Given a marked E-tableau T and a non-leaf node N with literal L which dominates marked leaves only. Let M be the set of nodes which are markings $\mu(N_i)$ of the leaf nodes N_i dominated by N. Obtain the set M' from M by removing N and all nodes dominated by N; note that all nodes in M' are on the path from the root to N, excluding the latter. Now, either M' contains solely the root node, in which case the label of the root is changed by adding the literal $\sim L$. Or M' contains an inner node N' which is dominated by all other nodes; then the label of the edge leading to N' is changed by adding $\sim L$.[7]

Additionally, the reduction rule has to be extended, as follows.

Procedure 5.2 (E-tableau reduction) Given a marked E-tableau T, select a leaf node N with literal L, then,

1. either select a dominating node N' with literal $\sim K$ and a most general unifier σ for L and K, and mark N with N',

2. or select a dominating edge or the root E with $\sim K$ in $\lambda(E)$ and a most general unifier σ for L and K; then mark N with the node immediately below the edge or with the root, respectively.

Finally, apply the substitution σ to the literals in the tableau.

The *tableau* and the *connection tableau calculus with folding up* result from the ordinary versions by working with edge-labelled tableaux, adding the folding up rule, substituting the old reduction rule by the new one, starting with a root labelled with the empty list, and additionally labelling all newly generated edges with the empty list.

For connection tableaux, the folding up rule is properly stronger concerning deductive power than the factorization rule.

Proposition 5.1 *Connection tableaux with factorization cannot polynomially simulate connection tableaux with folding up.*

Example 5.1 The class of clause sets S^n_m consisting of the clauses

$$
\begin{array}{ll}
\neg p_0, & \\
p_0 \vee \neg p^1_1 \vee \cdots \vee \neg p^1_m, & \\
p^1_i \vee \neg p^2_1 \vee \ldots \vee \neg p^2_m, & \text{for } 1 \leq i \leq m, \\
\qquad \cdots & \\
p^{n-1}_i \vee \neg p^n_1 \vee \cdots \vee \neg p^n_m, & \text{for } 1 \leq i \leq m, \\
p^n_1, & \\
\cdots & \\
p^n_m. &
\end{array}
$$

[7]The position of the inserted literal exactly corresponds to the *C-point* in the terminology used in [Shostak, 1976].

Proof We use the class defined in Example 5.1. It can easily be recognized that any closed connection tableau for S with top clause $\neg p_0$ has $\sum_{i=0}^{n} m^i$ leaf nodes. Also, factorization is not possible if we start with the top clause $\neg p_0$, since no two subgoals N_1, N_2 with identical literals occur with N_2 being a brother node of an ancestor of N_1. Therefore, the example class is intractable for connection tableau with factorization, for this specific top clause. However, there are linear connection proofs with factorization if one of the clauses

$$p_i^{n-1} \vee \neg p_1^n \vee \cdots \vee \neg p_m^n, \ 1 \leq i \leq m$$

is taken as top clause. In order to obtain an unsatisfiable class which is intractable for *any* selection of the top clause, we can apply the following trick—let us call it the *augmentation trick*. We modify the class given in Example 5.1 by adding a literal $\neg p_0'$ to the top clause $\neg p_0$, and by adding renamed copies of the other clauses where the new literals are all consistently renamed and distinct from the old ones. For the resulting formula it does not matter with which clause we start, since now in *any* construction of a closed connection tableaux with factorization inevitably either $\neg p_0$ or $\neg p_0'$ must occur as a subgoal. And in the proof of this subgoal no factorization steps are possible, so that the resulting closed subtableau is isomorphic to the large one for the old formula class. Consequently, the new example class is intractable for connection tableau with factorization. Both the formula class from Example 5.1 and the modified class have linear proofs for connection tableau with folding up, as shown in Figure 8 for the initial class with $\neg p_0$ as top clause; for reasons of brevity, in the figure, the presentation framework of connection matrices is used. The displayed proof needs $1 + m + n + (n-1)(m^2-1)$ inference steps, while the number of literal occurrences in the respective clause set is $1 + m + 1 + (n-1)(m+1)m + n = 2 + m + n + (n-1)(m^2+m)$. □

Conversely, the polynomial simulation in the other direction is possible, for a certain class of selection functions.

Proposition 5.2 *For arbitrary depth-first selection functions, (connection) tableaux with folding up linearly simulate (connection) tableaux with factorization.*

Proof Given any closed (connection) tableau T with factorization, let $\prec$ be its factorization dependency relation. By the strong node selection independency of factorization (Proposition 4.1), T can be constructed with any selection function. We consider a construction $S = (T_0, \ldots, T_m, T)$ of T with a depth-first selection function ϕ which respects the partial order of the factorization dependency relation $\prec$, i.e., for any two nodes N_1, N_2 in the tableau, $N_1 \prec N_2$ means that N_1 is selected before N_2; such a selection function exists since $\prec$ solely relates brother nodes. The deduction process S can be directly simulated by the (connection) tableau calculus with folding up, as follows. Using the same selection function ϕ, any expansion (extension) and reduction step in S is simulated by an expansion (extension) and reduction step. But, whenever a subgoal has been completely solved in the simulation, it is folded up. Since in the original deduction process, due to the pessimistic application of factorization, the factorization of a node N_1 with a node N_2 (with literal L) involves that N_2 has been solved before, in the simulation the literal L must have been folded up before. Now, *any* solved subgoal can be folded up at least

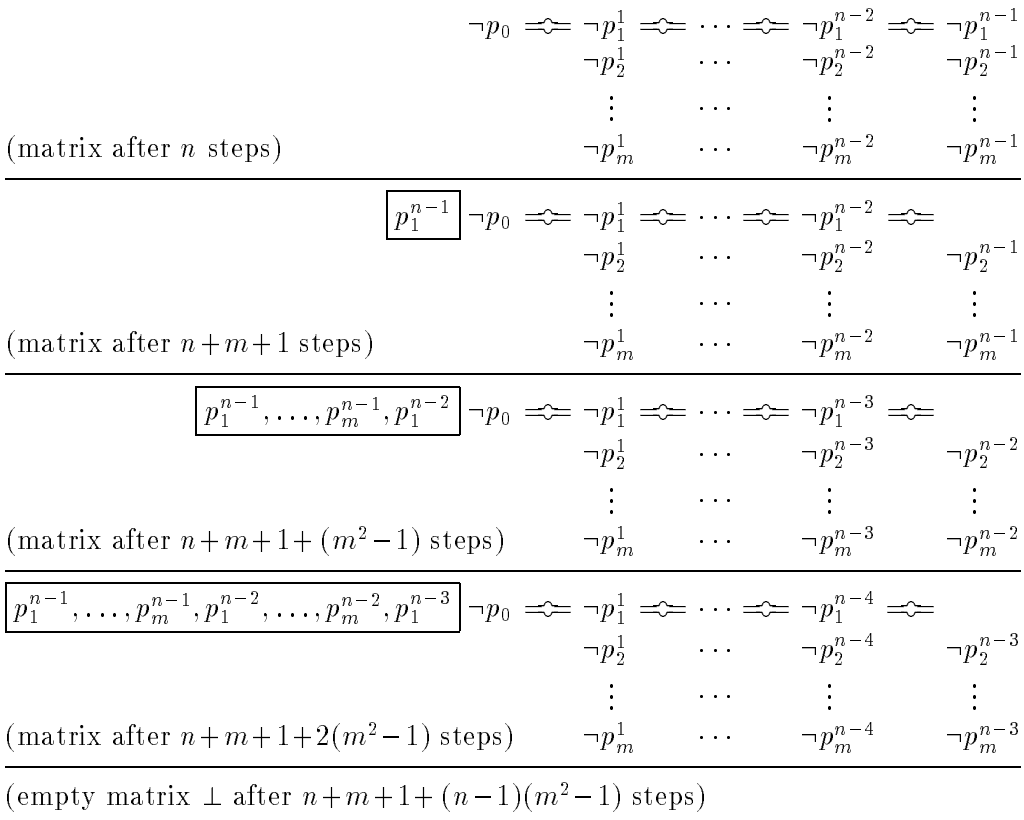

Figure 8: Linear connection proof with folding up for Example 5.1.

one step, namely, to the edge E above its predecessor (or the root). Since E (or the root) dominates N_1, the original factorization step can be simulated by a reduction step. The simulation of factorization by folding up is graphically shown in Figure 9. □

Finally, we show that the folding up rule, although strictly more powerful than factorization, is still a hidden version of the cut rule.

Proposition 5.3 *Tableaux with atomic cut and regular connection tableaux with atomic connection cut linearly simulate (connection) tableaux with folding up.*

Proof We perform the simulation proof for tableaux with cut. Given a tableau derivation with folding up, each folding up operation at a node N_0 adding the negation $\sim L$ of a solved subgoal L to the label of an edge above a node N (or to the root), can be simulated as follows. Perform a cut step at the node N with the atom of L as cut formula, producing two new nodes N_1 and N_2 labelled with L and $\sim L$, respectively; shift the solution of L from N_0 below the node N_1 and the part of the tableau previously dominated by N below its new successor node N_2; finally, perform a reduction step at the node N_0. Apparently, the unmarked branches of both tableaux can be injectively mapped to each other such that all pairs of corresponding branches contain the same leaf literals and the same sets of path literals, respectively. The simulation is graphically shown in Figure 10. □

In [Letz, 1993c] it is proven that connection tableaux with folding up linearly simulate tableaux with atomic cut (see also [Mayr, 1993]).

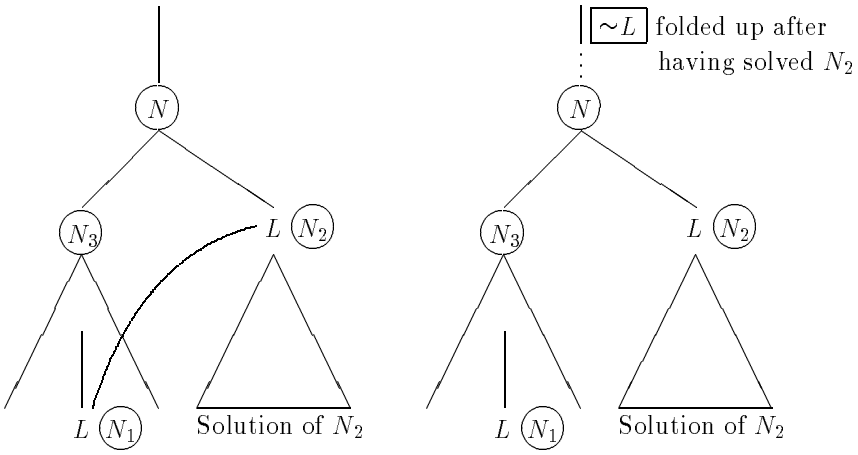

Figure 9: Simulation of factorization by folding up.

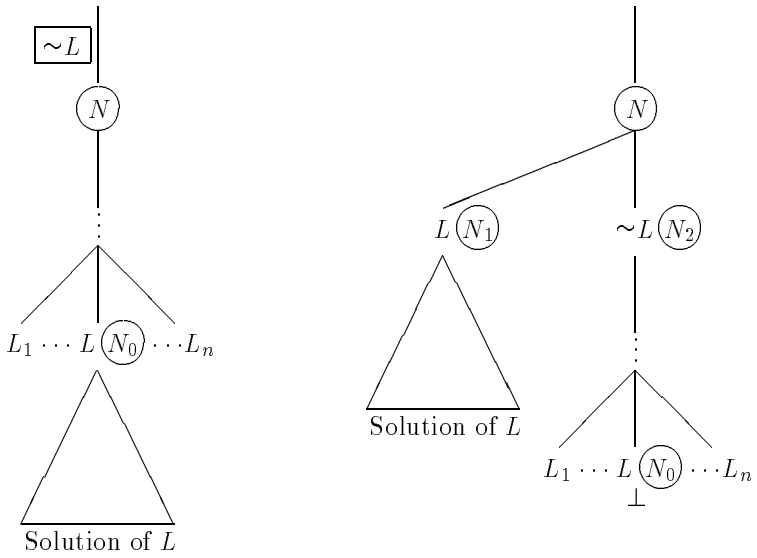

Figure 10: Simulation of folding up by cut.

6 The Folding Down Rule

The simulation of factorization by folding up also shows how a restriction of the folding up rule could be defined which permits an *optimistic* labelling of edges. If a strict linear (dependency) ordering is defined on the successor nodes $N_1, \ldots, N_m$ of any node, then it is permitted to label the edge leading to any node N_i, $1 \leq i \leq m$, with the set of the negations of the literals at all nodes which are smaller than N_i in the ordering. We call this operation the *folding down* rule (in German: "Umklappen"). The folding down operation can also be applied incrementally, as the ordering is completed to a linear one. The folding down rule is sound, since it can be simulated by the cut rule plus reduction steps, as illustrated in Figure 11. The rule can be viewed as a very simple and efficient way of *implementing* factorization. Over and above that, if also the literals on the edges are considered as path literals in the regularity test, an additional search space reduction can be achieved this way which is discussed in the next section. It should be noted that

it is very difficult to identify this refinement in the factorization framework. There, it is normally formulated in a restricted version, namely, as the condition that the set of literals at the *subgoals* of a tableau need to be consistent.

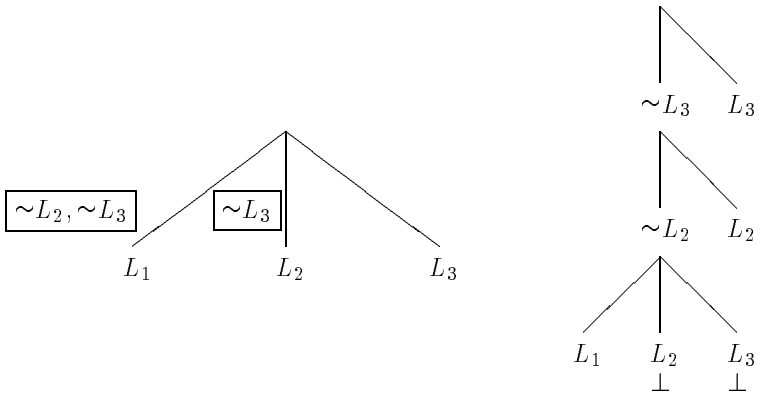

Figure 11: Simulation of folding down by cut.

Proposition 6.1 *(Regular) (connection) tableaux with folding down und (regular) (connection) tableaux with factorization linearly simulate each other.*

Note Folding down is essentially *Prawitz reduction* [Prawitz, 1960]. Hence, by the above proposition, Prawitz reduction and factorization have the same deductive power when added to the connection tableau calculus, and Prawitz reduction is properly weaker than folding up and atomic (connection) cut.

7 Enforced Folding Up and Strong Regularity

The folding up operation has been introduced as an ordinary inference rule which, according to its indeterministic nature, may be applied or not. Alternatively, we could have defined versions of the (connection) tableau calculi with folding up in which any solved subgoal *must* be folded up immediately after it has been solved. It is clear that whether folding up is performed freely, as an ordinary inference rule, or in an enforced manner, the resulting calculi are not different concerning deductive power, since the folding up operation is a monotonic operation which does *not decrease* the inference possibilities. But the calculi differ with respect to their search spaces, since by treating the folding up rule just as an ordinary inference rule, which may be applied or not, an additional and absolutely useless form of indeterminism is imported. Consequently, the folding up rule should better not be introduced as an additional inference rule, but as a tableau operation to be performed immediately after the solution of a subgoal. The resulting calculi will be called the *(connection) tableau calculi with enforced folding up.*

The superiority of the enforced folding up versions over the unforced ones also holds if the regularity restriction is added, according to which no two *nodes* on a branch can have the same literal as label. But apparently, the manner in which the folding up and the folding down rules have been introduced raises the question whether the regularity condition might be sharpened and extended to the consideration of the literals in the

labels of the edges, too. It is clear that such an extension of regularity does not go together with folding up, since any folding up operation makes the respective closed branch immediately violate the extended regularity condition. A straightforward remedy is to apply the extended condition to the *subgoal trees* of tableaux only.

Definition 7.1 (Strong regularity) An E-tableau T is called *strongly regular* if it is regular and on no branch of the subgoal tree of T a literal appears more than once, be it as a label of a node or within the label list of an edge or the root.

When the strong regularity condition is imposed on the connection tableau calculus with enforced folding up, then a completely new calculus is generated which is no extension of the regular connection tableau calculus, that is, not every proof in the regular connection tableau calculus can be directly simulated by the new calculus. This is because after the performance of a folding up operation certain inference steps previously possible for other subgoals may become impossible then. A folding up step may even lead to an immediate failure of the extended regularity test, as demonstrated below. Since the new calculus is no extension of the regular connection tableau calculus, we do not even know whether it is complete, since the completeness result for regular connection tableaux cannot be applied. In fact, the new calculus is *not complete* for arbitrary selection functions.

Proposition 7.1 *There is an unsatisfiable set S of ground clauses and a selection function ϕ such that there is no refutation for S in the strongly regular connection tableau calculus with enforced folding up.*

Example 7.1 The set S consisting of the clauses

$$\begin{array}{llll} \neg p \vee \neg s \vee \neg r, & p \vee s \vee r, & \neg q \vee r, & q \vee \neg r, \\ \neg p \vee t \vee u, & p \vee \neg t \vee \neg u, & \neg q \vee s, & q \vee \neg s, \\ & & \neg q \vee t, & q \vee \neg t, \\ & & \neg q \vee u, & q \vee \neg u. \end{array}$$

Proof Let S be the set of clauses given in Example 7.1, which is minimally unsatisfiable. The non-existence of a refutation with the top clause $p \vee s \vee r$ for a certain unfortunate selection function ϕ is illustrated in Figure 12. If ϕ selects the s-node, then two alternatives exist for extension, separated by a $\vee$. For the one on the left-hand side, if ϕ shifts to the p-subgoal above and completely solves it in a depth-first manner, then the enforced folding up of the p-subgoal immediately violates the strong regularity, indicated with a '$\lightning$' below the responsible $\neg p$-subgoal on the left. Therefore, only the second alternative on the right-hand side may lead to a successful refutation. Following the figure, it can easily be verified that for any refutation attempt there is a selection possibility which either leads to extension steps which immediately violate the old regularity condition or produce subgoals labelled with $\neg p$ or $\neg r$. In those cases, the selection function always shifts to the respective p- or r-subgoal in the top clause, solves it completely and folds it up afterwards, this way violating the strong regularity. Consequently, for such a selection function, there is no refutation with the given top clause. The same situation holds for any other top clause selected from the set. This can be verified in a straightforward though tedious manner. Alternatively, in order to shorten the proof, we may use the augmentation trick employed

in the proof of Proposition 5.1: by adding appropriate literals and clauses to the set one can easily obtain an input set in which the incompleteness result holds for *any* top clause. □

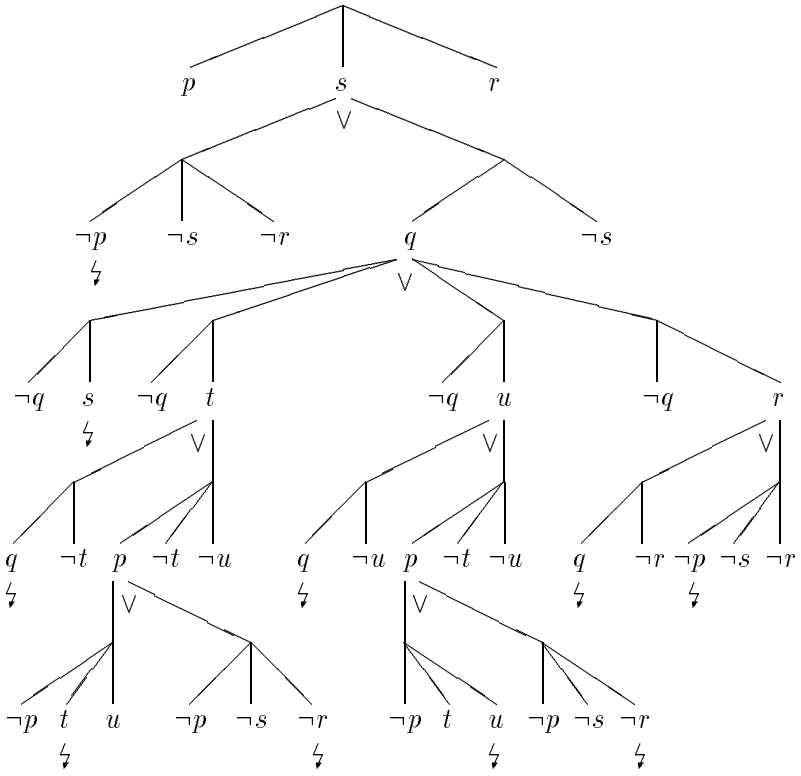

Figure 12: Incompleteness for free selection functions of the strongly regular connection tableau calculus with enforced folding up.

Note This result demonstrates that there is a trade-off between optimal selection functions and structural restrictions on tableaux. It would be interesting to investigate under which weakenings of the strong regularity the completeness for arbitrary selection functions might be obtained.

7.1 Completeness of the Calculus

If we restrict ourselves to depth-first selection functions, however, the new calculus is complete. In fact, something slightly stronger can be proven, using the following notions.

Definition 7.2 (Essentiality, relevance, minimal unsatisfiability) A formula F in a set of formulae S is called *essential in* S if S is unsatisfiable and $S \setminus \{F\}$ is satisfiable. An element F of a set of formulae S is named *relevant in* S if there exists an unsatisfiable subset $S' \subseteq S$ such that F is essential in S'. An unsatisfiable set of formulae S is said to be *minimally unsatisfiable* if each formula in S is essential in S.

It suffices to perform the completeness proof for the ground case, since the lifting to the first-order case is trivial, using Proposition 2.1.

Theorem 7.2 (Completeness for depth-first selection functions of strongly regular connection tableaux with enforced folding up) *For any finite unsatisfiable set S of ground clauses, any depth-first subgoal selection function, and any clause c_0 which is relevant in S, there exists a refutation of S with top clause c_0 in the strongly regular connection tableau calculus with enforced folding up.*

For the completeness proof we need an additional notion and a basic lemma.

Definition 7.3 (Strengthening) The *strengthening* of a set of clauses S *by* a set of literals $P = \{L_1, \ldots, L_n\}$, written $P \rhd S$, is the set of clauses obtained by first removing all clauses from S containing literals from P and afterwards adding the n unit clauses $L_1, \ldots, L_n$.

Lemma 7.3 (Strong Mate Lemma) *Let S be an unsatisfiable set of ground clauses. For any literal L contained in any relevant clause c in S there exists a clause c' in S such that*

(i) c' contains $\sim L$,

(ii) for every literal $L' \neq L$ in c': its complement $\sim L'$ is not contained in c, and

(iii) c' is relevant in the strengthening $\{L\} \rhd S$.

Proof From the relevance of c follows that S has a minimally unsatisfiable subset S_0 containing c; every formula in S_0 is essential in S_0. Hence, there is an interpretation[8] $\mathcal{I}$ for S_0 with $\mathfrak{I}(S_0 \setminus \{c\}) = \mathsf{true}$ and $\mathfrak{I}(c) = \mathsf{false}$, for the formula assignment $\mathfrak{I}$ of $\mathcal{I}$; $\mathfrak{I}$ assigns false to every literal in c. Define $\mathcal{I}' := (\mathcal{I} \setminus \{\sim L\}) \cup \{L\}$. Its assignment $\mathfrak{I}'$ maps c to true. The unsatisfiability of S_0 guarantees the existence of a clause c' in F_0 with $\mathfrak{I}'(c') = \mathsf{false}$. We prove that c' meets the conditions (i) – (iii). First, the clause c' must contain the literal $\sim L$, since otherwise $\mathfrak{I}(c') = \mathsf{false}$, which contradicts the selection of $\mathcal{I}$, hence (i). Secondly, for any other literal $L' \neq L$ in c': $\mathfrak{I}(L') = \mathfrak{I}'(L') = \mathsf{false}$. As a consequence, L' must not occur complemented in c, which proves (ii). Finally, the essentiality of c' in S_0 entails that there exists an interpretation $\mathcal{I}''$ with $\mathfrak{I}''(S_0 \setminus \{c'\}) = \mathsf{true}$ and $\mathfrak{I}''(c') = \mathsf{false}$, for the assignment $\mathfrak{I}''$ of $\mathcal{I}''$. Since $\sim L$ is in c', $\mathfrak{I}''(L) = \mathsf{true}$. Therefore, c' is essential in $S_0 \cup \{L\}$, and also in its subset $\{L\} \rhd S_0$. From this and the fact that $\{L\} \rhd S_0$ is a subset of $\{L\} \rhd S$ follows that c' is relevant in $\{L\} \rhd S$. □

Proof of Theorem 7.2 Let S be a finite unsatisfiable set of ground clauses and c_0 any relevant clause in S. We start with a tableau consisting simply of c_0 as top clause. We proof that for any subgoal N in the deduction process with a certain property there exists an inference step producing only subgoals with the same property. This inherited property is that the tableau clause determined by the position of N and its brother nodes is relevant in the strengthening of S by the extended path set[9] P from the root to N, excluding the latter. Suppose that N with literal L is such a subgoal with tableau clause c which is relevant in $P \rhd S$. If N is selected first for solution, either a reduction step can

[8] We view interpretations as sets containing, for any ground atom A constructible from the signature of a given formula, exactly one of the literals A or $\neg A$.

[9] The *extended path set* of a path contains all literals at the nodes of the path and in the lists at the edges and the root.

be performed at N, or, by the Strong Mate Lemma 7.3, a clause c' exists for an extension step which is relevant in $(P \cup \{L\}) \rhd S$, and we are done. Otherwise, brother subgoals of N might have to be solved first, leading to an increase of the context of N. The relevance of c in $P \rhd S$ entails the existence of a subset S' of S such that c is essential in $P \rhd S'$. Now we permit only solutions of the brother nodes of N using clauses from $S' \setminus \{c\}$ for extension; such solutions exist due to the completeness of the regularity restriction. By the soundness of the folding up rule, during such solutions of the brothers of N only literals can be inserted above N which are logically implied by the satisfiable set $P \rhd (S' \setminus \{c\})$. Consequently, L must be relevant in the increased context too, so that the second case can be reduced to the first one. The successful termination of any tableau construction procedure satisfying the mentioned properties follows from the relevance of the top clause c_0 in S and from the fact that for any input set only regular tableaux of finite depth exist. □

7.2 Combining Folding Up and Folding Down

The interesting question may be raised whether it is possible to combine the pessimistic folding up rule with the optimistically-oriented folding down rule. We explain the combination for depth-first selection functions. Whenever a subgoal is selected for solution, before the solution process is started, all other unsolved brother nodes are folded down to the edge leading to N. The additional literals on the edge increase the number of inference possibilities, but they also increase the possibilities for a failure of the strong regularity test, and hence achieve additional search pruning. A naïve combination of folding down with the folding up rule, however, immediately results in an unsound calculus, as illustrated in Figure 13 with a "refutation" of the satisfiable set of clauses

$$\{\neg p \vee \neg q, \neg p \vee q, \neg q \vee p\}.$$

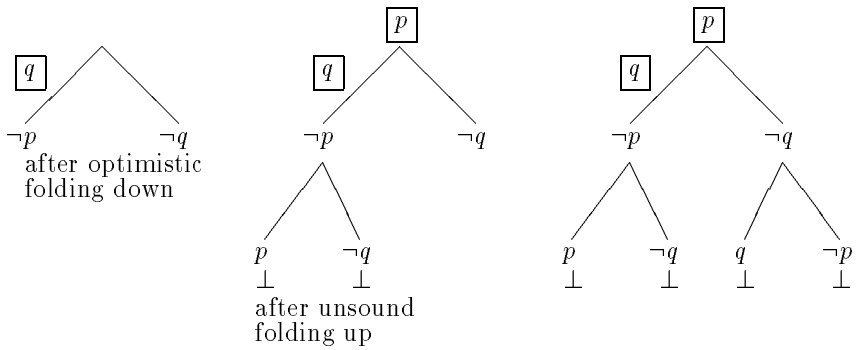

Figure 13: An unsound combination of folding up and folding down.

In the incorrect deduction, the $\neg p$-subgoal is selected first for solution. Before it is solved the unsolved $\neg q$-subgoal is folded down to the edge above the $\neg p$-subgoal. Then the latter is solved using the framed literal q. Thereupon, according to the way the reduction and folding up operations have been defined, the $\neg p$-subgoal may be folded up to the

root; this is the unsound operation. Afterwards, the $\neg q$-subgoal can be solved using the framed literal p.

We briefly sketch how a sound combination of folding up and folding down could be achieved. Apparently, the literals inserted into the labels of the edges by folding down operations need to be treated differently. The easiest solution would be to explicitly use the simulation of folding down by cut illustrated in Figure 11. In this simulation the dependency structure of the optimistic folding down rule is expressed in a pessimistic manner, which is compatible with the folding up rule.

8 Tableau Subsumption and Anti-Lemmata

In contrast to the most successful style of resolution theorem proving which is based on a formula enumeration or *saturation* procedure, such an approach is not possible in the connection tableau framework, because, unlike resolution and unlike the original tableau calculus, the connection tableau calculus is not *proof confluent*, that is, not every refutation attempt of an unsatisfiable formula can be completed successfully. This possibility of making irreversible decisions in the calculus demands a different organization of the proof process, namely, as a *deduction enumeration* instead of a formula enumeration procedure. The structure to be explored by a tableau proof procedure is the search tree induced by the underlying tableau calculus and the given selection function, which together constitute what we call a *determined* tableau calculus.[10]

Definition 8.1 (Determined tableau calculus) A *(selection) determined tableau calculus* is a pair $\langle C, \phi \rangle$ consisting of a tableau calculus C and a subgoal selection function ϕ.

Definition 8.2 ((Tableau) search tree) Let S be a set of clauses and $\mathcal{C} = \langle C, \phi \rangle$ a determined tableau calculus. The *(tableau) search tree of S and $\mathcal{C}$* is a tree $\mathcal{T}$ labelled with tableaux defined as follows.

1. The root of $\mathcal{T}$ is labelled with a tableau consisting of a root node only.
2. Every non-leaf node $\mathcal{N}$ in $\mathcal{T}$ labelled with a tableau T has as many successor nodes as there are successful applications of a single inference step in C applied to the subgoal N in T selected by ϕ, and the successor nodes of $\mathcal{N}$ in $\mathcal{T}$ are labelled with the respective resulting tableaux.

The leaf nodes of a (tableau) search tree that are labelled with solved tableaux are called *success nodes*.

8.1 Tableau Subsumption

There are two different methodologies for reducing the search effort of tableau enumeration procedures. On the one hand, one can attempt to *refine* the tableau calculus, that is, disallow certain inference steps if they produce tableaux of a certain *structure*—the connectedness and the regularity conditions are of this type. The effect on the tableau

[10] It is reasonable to fix the selection function, since otherwise one would have to search additionally through all possible selection functions.

search tree is that the respective nodes together with the dominated subtrees can be ignored, so that the branching rate of the tableau search tree decreases. These *structural* methods of redundancy elimination are *local* pruning techniques in the sense that they can be performed by looking at single tableaux only.

The other approach is to improve the *proof search procedure* so that information coming from the proof search itself can be used to even eliminate proof attempts not excluded by the calculus. More specifically, these *global* methods compare *competitive* tableaux in the search tree, i.e., tableaux on different branches, and attempt to show that one tableau (together with its successors) is redundant in the presence of the other. A natural approach here is to exploit *subsumption* between tableaux, in a similar manner subsumption between clauses is used in formula saturation procedures like resolution.

To this end, the notion of subsumption has to be generalized from clauses to literal trees. For an elegant definition of subsumption between literal trees the following notion of literal tree *contractions* proves helpful.

Definition 8.3 ((Literal) tree contraction) A (literal) tree T is called a *contraction* of a (literal) tree T' if T' can be obtained from T by attaching n (literal) trees to n non-leaf nodes of T, for some $n \geq 0$.

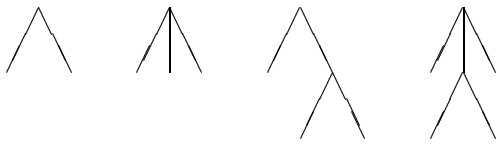

Figure 14: Illustration of the notion of tree contractions.

In Figure 14, the tree on the left is a contraction of itself, of the second and the fourth tree but not a contraction of the third one. No-one of the other trees is a contraction of another one.

Now, subsumption can be defined easily, by building on Definition 2.8 which determines when a literal tree is *more general* than a literal tree.

Definition 8.4 (Literal tree subsumption) A literal tree T *(strictly) subsumes* a literal tree T' if T is (strictly) more general than a contraction of T'.

Since the exploitation of subsumption between entire tableaux has not enough reductive potential, we favour the following form of subsumption deletion.

Procedure 8.1 (Subsumption deletion) For any pair of *competitive* nodes $\mathcal{N}$ and $\mathcal{N}'$ in a tableau search tree $\mathcal{T}$, if the subtree of the tableau at $\mathcal{N}$ subsumes the subtree of the tableau at $\mathcal{N}'$, then the whole subtree of the search tree with root $\mathcal{N}'$ is deleted from $\mathcal{T}$.

With subsumption deletion a form of *global* redundancy elimination is achieved which is complementary to the purely tableau *structural* pruning methods discussed so far.

In order to illustrate that cases of subgoal tree subsumption inevitably occur in the search for proofs, we will now present a phenomenon of logic which sheds light on a problematic property of proof search. Apparently, for the purposes of optimizing proof search

it is crucial to avoid as much redundancy as possible. Thus one should strive for identifying a minimally unsatisfiable subset of the input set under investigation, or, equivalently, a subset in which every relevant formula is essential. The problematic property of logic with respect to search pruning is that even if we would have identified a minimally unsatisfiable subset of an input set, the strengthening process (which is implicitly performed in tableau calculi) may introduce new redundancies. Let us formulate this more precisely.

Proposition 8.1 *If a set of clauses S is minimally unsatisfiable, and L is a literal occurring in clauses of S, then the strengthening $\{L\} \rhd S$ may contain more than one minimally unsatisfiable subsets, or, equivalently, not every relevant clause in $\{L\} \rhd S$ may be essential.*

Proof We use a set S consisting of the following propositional clauses

$$\begin{array}{llll} p \vee q, & \neg q \vee r, & \neg p \vee \neg q \vee \neg s, & \neg p \vee q \vee r, \\ p \vee \neg r \vee \neg s, & \neg q \vee s, & \neg p \vee \neg r \vee s, & \neg p \vee q \vee \neg r \vee \neg s. \end{array}$$

The set S is minimally unsatisfiable, but in the strengthening $\{p\} \rhd S$ the clauses $\neg q \vee r$ and $\neg q \vee s$ both are relevant but no more essential, since the new unit clause p is also falsified by the interpretations $\{\neg p, q, \neg r, s\}$ and $\{\neg p, q, r, \neg s\}$ which have rendered the two mentioned clauses essential in S, respectively. □

In more concrete terms, if we use the clause $p \vee q$ as top clause, then there are at least two different subproofs of the subgoal labelled with p. Consequently, there are at least two tableaux in the search tree of a respective determined tableau calculus whose subgoal trees subsume each other (in this case the subgoal trees are even identical). Since this type of redundancy can never be identified with tableau structure refinements like connectedness, regularity, or allies, we have uncovered a natural limitation of such purely local pruning methods.

8.2 Incompatibility of Subsumption with (Strong) Regularity

Similar to the case of resolution where certain refinements of the *calculus*, i.e., restrictions of the resolution *inference rule*, become incomplete when combined with subsumption deletion, such cases also occur for refinements of tableau calculi. Formally, the compatibility with subsumption deletion is the following property.

Definition 8.5 (Compatibility with subsumption (deletion)) A determined tableau calculus is said to be *compatible with (strict) subsumption (deletion)* if its search tree $\mathcal{T}$ has the following property. For arbitrary pairs of nodes $\mathcal{N}, \mathcal{N}'$ in $\mathcal{T}$, if the subgoal tree S of the tableau T at $\mathcal{N}$ (strictly) subsumes the subgoal tree S' of the tableau T' at $\mathcal{N}'$ and if $\mathcal{N}'$ dominates a success node, then $\mathcal{N}$ dominates a success node.

Apparently, the (connection) tableau calculus (with factorization, folding down, or folding up) is compatible with subsumption. But the integration of the full regularity condition poses problems.

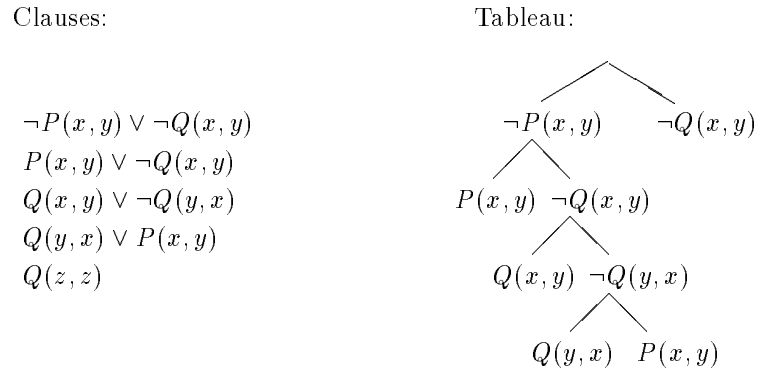

Figure 15: The incompatibility of subsumption and regularity.

Proposition 8.2 *The regular connection tableau calculus (with factorization, folding down, or folding up) is incompatible with subsumption.*

Proof We use the unsatisfiable set of clauses displayed on the left of Figure 15. Taking the first clause as top clause and employing the depth-first left-most selection function, the first subgoal N labelled with $\neg P(x,y)$ can be solved by deducing the tableau T depicted on the right of the figure. Since N has been solved optimally, i.e., without instantiating its literal, the subgoal tree of T subsumes the subgoal trees of all other tableaux working on the solution of N. Hence, all tableaux competing with T can be removed by subsumption deletion. But T cannot be extended to a solved tableaux, due to the full regularity condition—the crucial impediment being that an extension step into $Q(z,z)$ is not permitted, since it would render the already solved subtableau on the left irregular. To obtain a formula in which subsumption is fatal for *any* top clause formula, one can employ the augmentation trick used in the proof of Proposition 5.1. □

The apparent problem with full regularity is that it applies to entire tableaux whereas subsumption considers only their subgoal trees. A straightforward solution therefore is to restrict regularity to subgoal trees, too. The respective weakening of regularity is called *subgoal tree regularity.*

This is the first illustration of a conflict between refinements of the calculus and pruning of the proof procedure. Another more serious one is the combination of subsumption with strong regularity, since the new edge literals resulting from enforced folding up operations may disable the finding of subproofs for subsequent subgoals.

Proposition 8.3 *The strongly subgoal tree regular connection tableau calculus with enforced folding up is incompatible with subsumption.*

Proof We use the following slight modification of the set given in Example 7.1.

$\neg p \vee \neg s \vee \neg r,$	$\neg p'' \vee s \vee p' \vee r,$	$p'',$	$\neg q \vee r,$	$q \vee \neg r,$
$\neg p \vee \neg t \vee \neg u,$	$\neg p'' \vee t \vee p' \vee u,$	$\neg p' \vee p,$	$\neg q \vee s,$	$q \vee \neg s,$
$\neg p \vee s \vee r,$	$p \vee \neg s \vee \neg r,$		$\neg q \vee t,$	$q \vee \neg t,$
$\neg p \vee t \vee u,$	$p \vee \neg t \vee \neg u,$		$\neg q \vee u,$	$q \vee \neg u.$

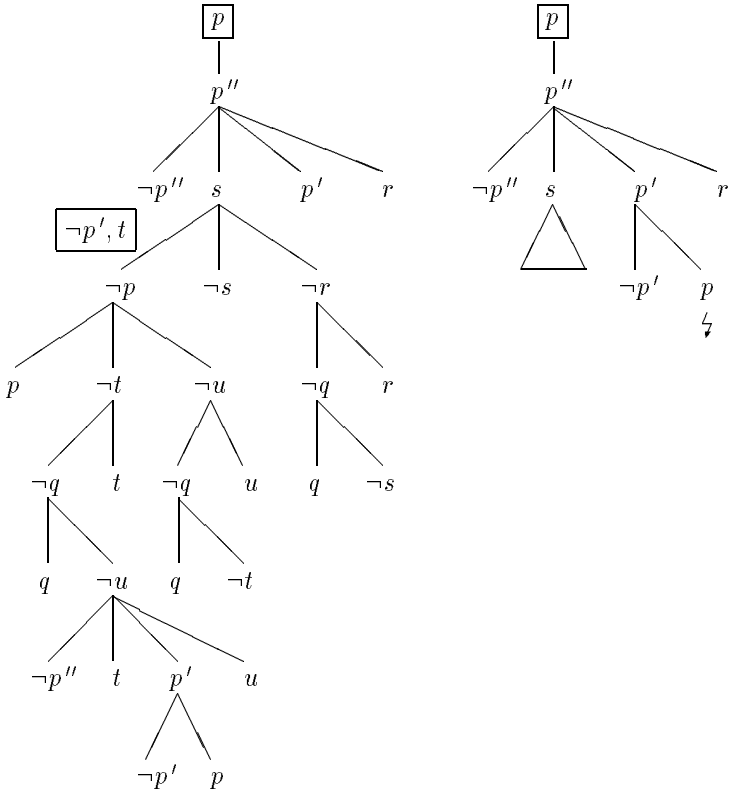

Figure 16: The incompatibility of subsumption and strong regularity.

The set is unsatisfiable (however, not minimally unsatisfiable). For the top clause p'', under a depth-first left-most selection function, one of the clauses $\neg p'' \vee s \vee p' \vee r$ or $\neg p'' \vee t \vee p' \vee u$ has to entered. If the first of these clauses is entered, the subgoal labelled with s can be solved in such a manner that the tableau T is deduced which is displayed on the left of Figure 16. In the proof process the literal $\neg p$ has to be folded up into the root (we fold up only if the literals to be inserted dominate non-empty subgoal trees). All tableaux competing with T in the solution of the s-subgoal can be ignored due to subsumption deletion. The next subgoal selected in T (the p'-subgoal), however, cannot be solved, since the only possible inference step, an extension step into the clause $\neg p' \vee p$, violates the strong regularity. Due to the symmetry in the formula—interchange s with t and r with u—the same failure of the search process can be constructed when the other possible clause $\neg p'' \vee s \vee p' \vee r$ is attached to the top subgoal labelled with p''. Hence no closed tableau can be found for the top clause p''. Since p'' is essential in the input set, a set can be constructed in which this failure occurs for *any* top clause, by applying the augmentation trick used in the proof of Proposition 5.1. □

Since there seems no reasonable compatible combination of subsumption with strong regularity, one has to decide which one of both pruning mechanisms should be applied in a proof procedure.

8.3 Subsumption Deletion in Tableau Search Procedures

There is the following additional completeness problem with subsumption deletion. Even for determined tableau calculi that are compatible with subsumption there exist search trees $\mathcal{T}$ in which, for any node $\mathcal{N}'$ dominating a success node, there is a competitive node $\mathcal{N}$ with a greater depth in $\mathcal{T}$ such that the subtree of the tableau at $\mathcal{N}$ subsumes the subtree of the tableau at $\mathcal{N}'$. Then an unfortunate proof search procedure could always delete the nodes with smaller depths in favour of the ones with greater depths and this way never arrive at a success node. A simple formula illustrating this phenomenon is given in Example 8.1. The search tree with the first clause as top clause of a tableau calculus with depth-first left-most selection function is shown in Figure 17, using the subgoal formula representation of subgoal trees.

Example 8.1 The set of the four clauses $\neg P(x) \vee \neg Q$, $P(x_1) \vee \neg P(x_2)$, $P(a)$, and Q.

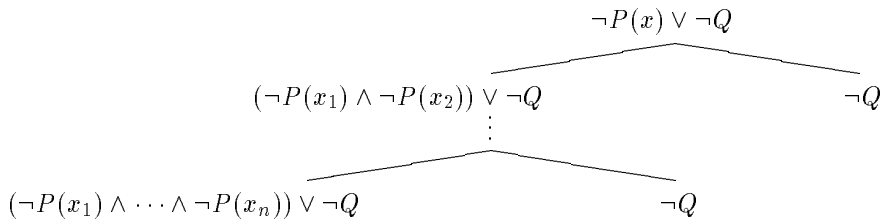

Figure 17: Structure of a subgoal formula search tree for Example 8.1.

There are the following two approaches to avoid such cases of incompleteness.

1. Perform subsumption deletion only if the *resources* to derive a subsuming subgoal tree are not greater than the resources for deriving the subsumed one.

2. Perform subsumption deletion only in *proper* cases of subsumption, i.e., when the subsuming subgoal tree is not itself subsumed by the subsumed one.

8.4 Anti-Lemmata

The observation that cases of subsumption inevitably will occur in practice motivates to organize the enumeration of tableaux in such a way that cases of subsumption can really be detected. This could be achieved with a proof procedure which explicitly constructs competitive tableaux and thus investigates the search tree in an breadth-first manner. The *explicit* enumeration of tableaux or subgoal trees, however, suffers from two severe disadvantages. The first one is also present in the standard resolution procedures, namely, the fact that, due to the extreme branching rate of the search tree, very quickly the available memory on a computer is exhausted; for tableaux or subgoal trees, which are much more complex structures than clauses, an explicit enumeration procedure may even be practically impossible. The second disadvantage is that the cost for adding new tableaux or subgoal formulae significantly increases during the proof process as the sizes of the proof objects increase, which is not the case for resolution procedures. These weaknesses

give sufficient reason why in practice no-one has seriously pursued an explicit tableau enumeration approach up to now.

The customary and successful paradigm therefore is to perform tableau enumeration in an implicit manner, using *consecutively bounded depth-first iterative deepening search* procedures. In this approach iteratively larger finite initial segments of a search tree $\mathcal{T}$ are explored, by imposing *completeness bounds*[11] on the structures of the permitted tableaux. Due to the construction process of tableaux from the root to the leaves, many tableaux have identical or structurally identical subparts. This motivates to explore finite initial segments in a *depth-first* manner, by strongly exploiting *structure sharing* techniques and *backtracking*. Accordingly, at each time only one tableau is in memory, which is extended following the branches of the search tree, and truncated when a leaf node of the respective inital segment of the search tree is reached. Although according to this methodology initial parts of the search tree are explored multiply, no significant efficiency is lost if the initial segments increase exponentially [Korf, 1985]. The advantage is that, due to the application of Prolog techniques, extremely high inference rates can be achieved this way (see [Stickel, 1988] and [Letz et al., 1992]).

The disadvantage of this approach, however, is that it is not easily possible to implement subsumption techniques in an adequate way. A restricted concept of subsumption deletion, however, can be achieved with the mechanism of so-called *anti-lemmata*. Here, we present the technique for depth-first selection functions only.

Definition 8.6 (Solution of a node, anti-lemma for a node) Given a tableau search tree $\mathcal{T}$ for a determined tableau calculus with a depth-first selection function. Let $\mathcal{N}$ be a node in $\mathcal{T}$, T the tableau at $\mathcal{N}$, N the selected subgoal in T, and S the set of subgoals of T. If $\mathcal{N}'$ is a node in $\mathcal{T}$ dominated by $\mathcal{N}$ such that the set of subgoals of the tableau at $\mathcal{N}'$ is $S \setminus \{N\}$, then the composition $\sigma = \sigma_1 \cdots \sigma_n$ of substitutions applied to the tableaux on the way from $\mathcal{N}$ to $\mathcal{N}'$ is called a *solution of N at $\mathcal{N}$ via $\mathcal{N}'$*.
If $\mathcal{T}'$ is an initial segment of $\mathcal{T}$ containing the nodes $\mathcal{N}'$ (and hence $\mathcal{N}$) such that neither $\mathcal{N}'$ is itself a success node nor dominates a success node in $\mathcal{T}'$, then the solution σ is named an *anti-lemma for N at $\mathcal{N}$ in $\mathcal{T}'$*.

We describe how anti-lemmata are applied in a search procedure which explores tableau search trees in a depth-first manner using backtracking.

Procedure 8.2 (Generation, application, deletion of an anti-lemma)
Let $\mathcal{T}'$ be a finite initial segment of a tableau search tree $\mathcal{T}$.
Whenever a subgoal N in a tableau at a node $\mathcal{N}$ in $\mathcal{T}'$ has been solved via a node $\mathcal{N}'$, then the computed solution σ is stored at the node N. If $\mathcal{N}'$ dominates no success node in $\mathcal{T}'$ and the proof procedure backtracks over the node $\mathcal{N}'$, then σ is turned into an anti-lemma.
In any alternative solution process of the subgoal N below the node $\mathcal{N}$, if a substitution $\tau = \tau_1 \cdots \tau_m$ is computed such that one of the anti-lemmata stored at the node N is more general than τ, the proof procedure immediately backtracks. For all nodes below $\mathcal{N}$ in

[11] Mathematically, a *completeness bound* can be viewed as a functional assigning to any search tree $\mathcal{T}$ a mapping $\mathcal{B}_\mathcal{T}$ from the set of natural numbers into the collection of initial segments of $\mathcal{T}$ such that, for any node $\mathcal{N}$ in $\mathcal{T}$, there exists an n with $\mathcal{N}$ in $\mathcal{B}_\mathcal{T}(n)$.

which N is solved, the anti-lemmata at N are ignored.
When the node $\mathcal{N}$ at which N was selected for solution is backtracked, then all anti-lemmata at N are deleted.

Apparently, whenever an anti-lemma σ for a tableau node N in a tableau T is more general than a substitution τ computed during an alternative solution attempt of N, then the subgoal tree of T strictly subsumes the subgoal tree of the alternative tableau. Consequently, the anti-lemma mechanism achieves a restricted form of subsumption deletion.

8.5 Comparison with other Techniques

The *caching* technique proposed in [Astrachan and Stickel, 1992] suggests to record the solutions of subgoals independently of the contexts in which the subgoals appear. Then, cached solutions can be used for solving subgoals by *lookup* instead of search. In the special case in which no solutions for a cached subgoal exist, the cache acts in the same manner as the mechanism of anti-lemmata. One difference is that anti-lemmata take the path context into account and hence are compatible with subgoal tree regularity whereas the mentioned caching technique is not. On the other hand, permanently cached subgoals without context have more cases of application than the temporary and context-dependent anti-lemmata. The main disadvantage of the context-ignoring caching technique, however, is that its applicability is restricted to the Horn case.

Note that the first aspect of the mentioned caching technique, namely, replacing search by lookup, cannot be captured with a temporary mechanism like anti-lemmata, since lookup is mainly effective for *different* subgoals whereas anti-lemmata are used on different solutions of *one and the same* subgoal.

9 Implementation

We have implemented the new inference mechanisms on top of version V3.1 of the model elimination theorem prover SETHEO [Letz et al., 1992]. The actual version V3.1 differs from the one described in the paper in the subsequent two respects.

Integration of anti-lemmata On the one hand, the following restricted but very efficient form of the anti-lemma mechanism has been integrated in the system. The solution σ of a subgoal N at a node $\mathcal{N}$ via a node $\mathcal{N}'$ is stored as an anti-lemma for N only if the proof procedure backtracks from $\mathcal{N}'$ through to the node $\mathcal{N}$ without performing alternative inferences in the meantime. This facilitates that solutions of subgoals need not be stored at all, since solution substitutions can be directly converted into anti-lemmata, which can be done *without copying* structures.

Integration of a constraint handler In [Letz et al., 1992] it is described how the applied tableau pruning techniques (in cases of irregularity, tautology and subsumption of tableau clauses) can be implemented efficiently by using *syntactic disequation constraints* between lists of terms. While in the old version of SETHEO it was not possible to check the respective constraints after each inference step, a constraint mechanisms has been

integrated into the actual version which controls the generation, update and satisfaction of the disequation constraints. Since the constraint handler is a subroutine of the unification procedure, the full tableau pruning is achieved now.

Implementation of the new inference rules The enforced folding up mechanism and the optimistic factorization rule (folding down) could be fully integrated into the abstract machine of SETHEO. Since all needed data structures were already present in the system, only minor changes had to be made—to give an example, the folding up operation could be integrated with adding about 20 lines of C code. Also, due to the depth-first selection function used in SETHEO, the identification of the folding up point (the C-point) could be further optimized, as follows. In each reduction step into a predecessor literal the *selection time* of the closed subgoal is noted at the literal occurrence, where greater values overwrite smaller ones. This facilitates that when a literal L of a solved node N with generation time t is being folded up, one simply has to walk towards the root starting from the predecessor node of N until a predecessor literal (occurrence) is reached which is marked with a selection time greater than t; the insertion point (C-point) of $\neg L$ is immediately above this position. Furthermore, it is important to emphasize that not a *copy* of the literal has to be inserted but simply a *pointer* to the literal. This is the main implementational advantage of folding up if compared with a full lemma mechanisms including copying and renaming of clauses. As a consequence, the implementational overhead of the additional operations is almost negligible. To our knowledge, the resulting system represents the first *professional* implementation of C-reduction [Shostak, 1976].

10 Experiments

The compatibility results between the different inference and pruning mechanisms have naturally led us to the comparison of four complete proof procedures, one cut-free, one with factorization, and two with folding up.

1. The cut-free reference point is version V3.1 of SETHEO as described above, with anti-lemmata and with regularity restricted to subgoal trees.

2. In the second system we use optimistic factorization (folding down) with anti-lemmata and strong subgoal tree regularity.

3. The third system performs enforced folding up with strong regularity (in its full form) but no anti-lemmata.

4. Finally, in the fourth system we apply enforced folding up with anti-lemmata and with regularity restricted to subgoal trees.

We have tested the proof procedures on a sample of 12 formulae ranging from problems which are not too simple up to problems which are very hard for cut-free pure top-down oriented model elimination procedures. We have selected examples from the collection [Wilson and Minker, 1976] and the intermediate value theorem (ivt) communicated by

D. W. Loveland. In order to make absolutely clear which formulae we have taken—a lot of modifications are around in the literature—we have made the exact problem formulations available via ftp.[12] The problems were run on a SUN SPARC 10.

10.1 Search Strategy and Completeness Bounds

The employed search strategy is consecutively bounded iterative deepening search. Under this strategy, the simplest and most widely used completeness bounds are the following two. The first one limits the search in an iteration i to tableaux with *depth* i; more precisely, all tableaux are excluded that have nodes of a depth $> i$, not considering tableau unit clauses. With the second bound the *inferences* are restricted for the solution of tableaux, which, for connection tableau, is exactly the number of leaf nodes of a tableau (if we do not count the start step). In [Letz et al., 1992] results of an experimental comparison between both search modes are given.

Compatibility problems of the bounds with subsumption deletion While an iterative-deepening proof procedure using the depth bound is fully compatible with anti-lemmata and subsumption deletion, with the inference bound cases of incompleteness may occur, of the type mentioned in Subsection 8.3. In order to avoid them, we apply the first solution mentioned there, namely, perform failure due to anti-lemmata only if the ressources to derive the subsuming subgoal tree are not greater than the ones for deducing the subsumed one.

Unfortunately, each of the two bounds has severe disadvantages: in a word, the depth bound does not sufficiently restrict the breadth of tableaux while the inference bound does not sufficiently limit the depth. This has motivated us to use a *combination* of the two bounds. Experimental results suggest that it is not so important to find the *optimal* combination between both bounds—it may not exist anyway—it seems that *any* reasonable combination statistically performs better than both of the pure bounds separately. In order to illustrate this we have used two different combinations. Moreover, the bounds we employ are not *fixed* combinations of depth and inferences, but consider a bit of the structure of the underlying formula, by taking the average number of literals per clause into account.

The applied ressource bounds For the combination of the bounds, the depth bound is taken as a basis. Starting with tableaux of depth 3, we iterate by incrementing the depth by 1. The additional inference limitation for an iteration with depth k is computed as follows. Let ℓ be the average clause length in the given formula.

1. The first combined bound sets the inferences to $(k-1)^\ell$.
2. The second combined bound sets the inferences to $\frac{k^\ell}{2}$.

Example 10.1 Consider problem ex5, a non-Horn formula with 15 clauses and average clause length 2.00. The iteration sequences of the two bounds are the following.

[12] 'ftp 131.159.8.35' or 'ftp flop.informatik.tu-muenchen.de', login as 'anonymous', 'cd pub/ki/formulae'.

depth	3	4	5	6	7	...
inferences (Bound 1)	4	9	16	25	36	...
inferences (Bound 2)	5	8	13	18	25	...

Using both bounds the problem can be proven by the first proof procedure, the cut-free SETHEO V3.1. In either case a proof is found with depth 6. The total iterated run times are 11.4 seconds for Bound 1 and 2.4 seconds for Bound 2. The formula ex5 is a good illustration of the superiority of combined bounds, since the problem cannot be solved with one of the pure bounds within two hours.

10.2 Results of the Measurements

The results for the 12 examples are displayed in the following table. Since we did not want to spend two lines per problem, we have run the two mentioned bounds in parallel, in competitive mode. The run-time given in the table then is the minimum of the run-times of both bounds times 2 (in seconds). Also, this way a more *robust* proof procedure is obtained which is not so sensitive to the selection of the right bound. The values in the 'proof'-column are the numbers of proof inferences (ignoring the start steps), in brackets the numbers of factorization steps and reduction steps into framed literals are given, respectively.

Table of experiments

Problem			Model Elimination + anti-lemm.		+ fold. down + strong reg. + anti-lemm.		+ fold. up + strong reg. − anti-lemm.		+ fold. up − strong reg. + anti-lemm.	
Name	Horn	Size	time	proof	time	proof	time	proof	time	proof
wos1	yes	17	1.8	10	.2	11(2)	.1	11(1)	.1	11(1)
wos10	yes	20	25.6	15	68.3	15(0)	41.4	15(0)	48.5	15(0)
wos11	yes	22	4.5	9	9.1	9(0)	6.7	8(0)	6.5	8(0)
wos22	yes	34	3.8	14	11.8	14(0)	10.1	14(0)	11.5	14(0)
wos31	no	23	$\gg 10^4$		1214.8	55(16)	14.5	45(7)	12.5	55(10)
wos33	no	26	$\gg 10^4$		$\gg 10^4$		208.8	35(7)	37.5	59(18)
apabhp	no	18	130.7	14	149.7	14(0)	381.8	14(1)	253.9	14(1)
ex5	no	15	4.8	18	6.7	15(1)	7.3	14(1)	6.6	14(1)
ls108	no	16	17.8	40	8.4	23(3)	.4	21(3)	.2	21(3)
ls112	no	23	28.8	56	2.7	60(8)	2.1	60(9)	1.7	60(9)
ls121	no	21	5.7	27	32.4	29(0)	.3	20(3)	.3	20(3)
ivt	no	17	6787.6	99	9849.4	99(0)	15.2	41(13)	14.3	41(13)

Assessment of the results

The experiments clearly demonstrate that the proof procedures with folding up perform significantly better than the other two, with the fourth system being slightly more successful than the third one. Thus the intermediate value theorem can be solved easily as well as the problems wos31 and wos33 which are even more difficult for cut-free tableau systems. Note also that even in those four cases (wos10, wos11, wos22, apabhp) in which no reduction steps into edge literals are beneficial and the edge literals only introduce

redundancy, the *relative loss* of efficiency is limited to a factor of 3, if considering the fourth proof procedure. The *relative gain* in efficiency with respect to the other systems, however, is often by magnitudes.

Let us describe the benefit of shortening proofs for the proof *search* with the intermediate value theorem (ivt). The cut-free proof we have found needs 99 inference steps and has depth 8. With the folding up mechanism the proof length can be reduced to 41 steps and, which seems even more crucial, the depth of the proof to 6. Consequently, the proof is found two levels earlier. This explains the achieved speed-up of about 500. The same holds for the problems wos31, wos33, ls108, and ls121. Interestingly, even in cases where the proof is on the same level, a speed-up can be achieved, namely, for the problems wos1 and ls112.

Additional experiments have shown that indeed techniques like anti-lemmata or strong regularity are needed in order to successfully compensate the redundancies caused by the new inference rules folding up and folding down. Just to give one example, the solution process of the problem wos31 needs 40 times more time if neither strong regularity nor anti-lemmata are used.

11 Conclusion and Further Research

We have presented a thorough theoretical study of different additional inference rules for (connection) tableau calculi, like factorization, atomic cut, C-reduction and folding up, and Prawitz reduction and folding down. The uniform formulation of those techniques in the tableau framework has helped to clarify the relations and (polynomial) simulation possibilities between the different inference mechanisms. Since all of the techniques can be linearly simulated by the atomic cut rule, the shortening of proofs achieved with the rules is limited.[13] On the other hand, the favoured mechanisms can be implemented very efficiently. Experiments with a number of representative examples have demonstrated that even with the relatively weak (single-instance context-unit) lemmata used in the folding up technique, a significant shortening of proofs can be achieved which has dramatic consequences on the proof search.

For the future, we see the following interesting research perspectives. First, the method of *temporary* anti-lemmata should be extended to *permanent* anti-lemmata including path contexts; this way a synthesis of our pruning approach with the caching technique in [Astrachan and Stickel, 1992] could be achieved. Secondly, more sophisticated completeness bounds and iterative deepening techniques should be studied. Finally, it seems worthwhile to investigate more powerful methods of supporting the pure top-down oriented tableau procedures with permanent and renamable lemmata, which could be generated in a preprocessing phase or in specific subprocedures.

References

[Astrachan and Loveland, 1991] O. W. Astrachan and D. W. Loveland. METEORs: High Performance Theorem Provers using Model Elimination. In R. S. Boyer (ed.), *Automated Reasoning: Essays in Honour of Woody Bledsoe*. Kluwer Academic Publishers, 1991.

[13] For example, the *full* deductive power of resolution cannot be reached.

[Astrachan and Stickel, 1992] O. W. Astrachan and M. E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. *Proceedings of the 11th Conference on Automated Deduction (CADE-11)*, LNAI 607, Saratoga Springs, pages 224–238, Springer, 1992.

[Beth, 1955] E. W. Beth. Semantic Entailment and Formal Derivability. *Mededlingen der Koninklijke Nederlandse Akademie van Wetenschappen*, 18(13):309–342, 1955.

[Beth, 1959] E. W. Beth. *The Foundations of Mathematics*. North-Holland, Amsterdam, 1959.

[Bibel, 1981] W. Bibel. On Matrices with Connections. *Journal of the ACM*, 28:633–645, 1981.

[Eder, 1991] E. Eder. Consolution and its Relation with Resolution. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, pages 132–136, Morgan Kaufmann, 1991.

[Bibel, 1987] W. Bibel. *Automated Theorem Proving*. Vieweg Verlag, Braunschweig, second edition, 1987.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935. Engl. translation in [Szabo, 1969].

[Korf, 1985] R. E. Korf. Depth-First Iterative Deepening: an Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97–109, 1985.

[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):183–212, 1992.

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

[Letz, 1993b] R. Letz. The Deductive Power of the Cut Rule. Technical report, Technische Universität München, 1993.

[Letz, 1993c] R. Letz and K. Mayr. The Relation of Extended Tableau Systems with Semantic Trees. Technical report, Technische Universität München, 1993.

[Loveland, 1968] D. W. Loveland. Mechanical Theorem Proving by Model Elimination. *Journal of the Association for Computing Machinery*, 15(2):236–251, 1968.

[Loveland, 1969] D. W. Loveland. A Simplified Format for the Model Elimination Theorem-Proving Procedure. *Journal of the Association for Computing Machinery*, 16:349–363, 1969.

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

[Mayr, 1993] K. Mayr. Refinements and Extensions of Model Elimination. Proceedings of the *4th International Conference on Logic Programming and Automated Reasoning (LPAR'93)*, LNAI 698, St. Petersburg, pages 217–228, 1993.

[Prawitz, 1960] D. Prawitz. An Improved Proof Procedure. *Theoria*, 26:102–139, 1960.

[Reckhow, 1976] R. A. Reckhow. *On the Lenghts of Proofs in the Propositional Calculus*. PhD thesis, University of Toronto, 1976.

[Shostak, 1976] R. E. Shostak. Refutation Graphs. *Artificial Intelligence*, 7:51–64, 1976.

[Smullyan, 1968] R. M. Smullyan. *First Order Logic*. Springer, 1968.

[Stickel, 1988] M. A. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[Wilson and Minker, 1976] G. A. Wilson and J. Minker. Resolution, Refinements, and Search Strategies: a Comparative Study. *IEEE Transactions on Computers*, C–25:782–801, 1976.