Linear Logic with Isabelle: pruning the proof search tree

Philippe de Groote

INRIA-Lorraine – CRIN – CNRS
615 rue du Jardin Botanique - B.P. 101
F 54602 Villers-lès-Nancy Cedex – FRANCE
e-mail: Philippe.de.Groote@loria.fr

Abstract. This paper introduces a general backward proof search strategy for multiplicative additive linear logic. This strategy, which is based on Isabelle's basic tactics and tacticals, has been implemented and appears to be rather efficient. Its efficiency derives from several heuristics that we introduce in the paper. We prove that these heuristics preserve completeness.

1 Introduction

This paper relates an experiment in which we implemented linear logic [5] with Isabelle [9, 8].

Isabelle, a descendant of LCF, is a generic theorem prover that allows proofs to be constructed by backward chaining. In [10], Tammet introduces different backward and forward proof search strategies for linear logic. Tammet's work and experiments seem to show that forward search based on top-down resolution techniques is much more efficient than backward chaining. In contrast, our work here demonstrates that backward proof search competes with forward search when good criteria for pruning the proof search tree are used.

At the moment, our results concern only the constant-free fragment of multiplicative additive linear logic. Indeed we decided not to take exponentials into account for a first experiment in order to remain within the decidable fragment of propositional linear logic. The fact that we are restricting ourselves to a constant-free fragment is not that important. Taking the constants into account does not raise new difficulties except for the additive constant $\top$ (whose presence invalidates Proposition 6.1)

The paper is organised as follows. In the next section, we discuss the difficulties encountered when trying to implement a substructural logic such as linear logic in Isabelle. In particular we explain why a sequent-calculus implementation is mandatory.

In Section 3 we remind the reader of the basic tactics and tacticals of Isabelle that we use, in the sequel, to build our different proof search strategies.

Section 4 introduces a general proof search strategy based on the work of Galmiche and Perrier [2, 4]. This strategy, which is complete, appears to be rather inefficient.

In: P. Baumgartner, R. Hähnle and J. Posegga (Eds.), *Proceedings of the 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*,
Lecture Notes in Computer Science, 918, Springer-Verlag, (1995), pp. 263-277.

The remainder of the paper is dedicated to the development of heuristics that improve the efficiency of the general strategy without loosing completeness. Section 5 is concerned with the multiplicative fragment. We take the multiplicative additive fragment into account in Section 6.

In the last section, we discuss implementation issues and experimental results.

All along this paper, we assume that the reader is familiar with linear logic ([3, 11] are good introductions). In particular, Section 5 requires some knowledge of proof net theory. Some familiarity with Isabelle, while not assumed, may also be useful.

2 Specifying linear logic in Isabelle

Isabelle is a generic proof assistant. This means that Isabelle is not dedicated to a fixed logic but provides syntactic means to allow various logics to be defined. The family of logics currently available under Isabelle is rather broad [8]. This demonstrates that generic theorem proving is feasible but does not mean that Isabelle can easily accommodate any logic. In particular, substructural logics seem to fall outside the scope of Isabelle because:

- Isabelle's metalogic is a fragment of intuitionistic higher order logic,
- Isabelle is natural deduction oriented.

To see the problem, let us try to specify, in a natural way, intuitonistic linear implication ($\multimap$) in Isabelle. Possible introduction and elimination rules for implication are the following ([9]):

$$\bigwedge PQ.(\llbracket P \rrbracket \Rightarrow \llbracket Q \rrbracket) \Rightarrow \llbracket P \multimap Q \rrbracket \tag{1}$$

$$\bigwedge PQ.\llbracket P \multimap Q \rrbracket \Rightarrow (\llbracket P \rrbracket \Rightarrow \llbracket Q \rrbracket) \tag{2}$$

These two formulas correspond respectively to the deduction theorem and to the principle of *modus ponens*, which both hold for linear implication. But Formulas (1) and (2) also specify some sort of equivalence between object implication ($\multimap$) and meta-implication ($\Rightarrow$). More precisely, (1) and (2) imply that object implication inherits the properties of meta-implication. Since the latter corresponds to intuitonistic implication, it is not possible to give a natural deduction oriented specification of linear implication in Isabelle.

The solution to this problem is to define linear logic by specifying a sequent-calculus. To encode sequents in Isabelle is not too difficult. In particular, by taking advantage of higher-order unification, one may get associative unification for free [8]. However, with respect to classical or intuitionistic logic, linear logic presents one further difficulty: associative unification is not sufficient to get a system free of exchange rule. The set of rules that we have chosen is the following.

$$\vdash A^{\perp}, A \qquad \text{(Identity)} \qquad\qquad \frac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta} \qquad \text{(Cut)}$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, A \otimes B, \Delta} \text{ (Times)} \qquad \frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, A \parr B, \Delta} \text{ (Par)}$$

$$\frac{\vdash \Gamma, A, \Delta \qquad \vdash \Gamma, B, \Delta}{\vdash \Gamma, A \,\&\, B, \Delta} \text{ (With)}$$

$$\frac{\vdash \Gamma, A, \Delta}{\vdash \Gamma, A \oplus B, \Delta} \text{ (Plus−L)} \qquad \frac{\vdash \Gamma, B, \Delta}{\vdash \Gamma, A \oplus B, \Delta} \text{ (Plus−R)}$$

$$\frac{\vdash \Gamma, \Delta, A, \Lambda}{\vdash \Gamma, A, \Delta, \Lambda} \text{ (Perm−1)} \qquad \frac{\vdash \Gamma, A, \Delta, \Lambda}{\vdash \Gamma, \Delta, A, \Lambda} \text{ (Perm−2)}$$

Among these rules, the only one for which associative commutative unification would be needed is the introduction rule for $\otimes$. Consequently, the exchange rules (Perm–1) and (Perm–2) will be needed only when using the Rule (Times). As in [5], we do not have rules for implication and negation, these connectives being defined by the following equations:

$$A \multimap B = A^{\perp} \parr B$$

$$(A^{\perp})^{\perp} = A$$
$$(A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp} \qquad (A \parr B)^{\perp} = A^{\perp} \otimes B^{\perp}$$
$$(A \,\&\, B)^{\perp} = A^{\perp} \oplus B^{\perp} \qquad (A \oplus B)^{\perp} = A^{\perp} \,\&\, B^{\perp}$$

At this point in the discussion, one may wonder if there is any interest in using Isabelle to implement linear logic. We have encountered two problems that enforce the following decisions:

- to encode a sequent-calculus,
- explicitely to use exchange rules in order to simulate commutative unification.

These two choices sound artificial and one could argue that Isabelle is simply not adapted to linear logic. However implementing linear logic with Isabelle offers two advantages:

- Isabelle's tactics are easy to develop and to adapt. Consequently, Isabelle is a good testbed for developing proof search strategies.
- In Isabelle, the axioms and inference rules of a logic are not wired in the proof search strategies. Consequently, the correctness of the tactics, for a given object logic, depends only of the soundness of the static definition the object logic. In other words, when developing a tactic, one has only to prove its completeness (its correctness being ensured by the system as long as the definition of the object logic is sound).

3 Isabelle's tactics and tacticals

The interest of Isabelle for developing proof search strategies is mainly due to the built-in notions of *tactic* and *tacticals* [8].

Roughly speaking, tactics are used to transform the current state of a backward proof into a new state. More precisely, tactics are functions from proof states to collections of proof states because the application of a given tactic may yield several possible results. This non-determinism, implemented using lazy evaluated sequences, allows Isabelle to backtrack.

One of the most basic tactics is the resolution tactic *resolve_tac*. This tactic refines the proof state by resolving one specified subgoal with a given inference rule. Consider, for instance the following subgoal:

$$\vdash a \parr b, a \parr b^{\perp}, a^{\perp} \parr b, a^{\perp} \parr b^{\perp} \tag{3}$$

To apply *resolve_tac* with Rule (Par) on this subgoal amounts to replace (3) by (4):

$$\vdash a, b, a \parr b^{\perp}, a^{\perp} \parr b, a^{\perp} \parr b^{\perp} \tag{4}$$

Applying *resolve_tac* on (3) yields also other possible results due to the associative unification:

$$\vdash a \parr b, a, b^{\perp}, a^{\perp} \parr b, a^{\perp} \parr b^{\perp}$$
$$\vdash a \parr b, a \parr b^{\perp}, a^{\perp}, b, a^{\perp} \parr b^{\perp}$$
$$\vdash a \parr b, a \parr b^{\perp}, a^{\perp} \parr b, a^{\perp}, b^{\perp}$$

These other results are the alternatives to (4) on which Isabelle can possibly backtrack.

Tacticals are composition operators for tactics. **THEN**, **ORELSE**, and **APPEND** are three typical tacticals, which correspond respectively to sequential composition, deterministic and undeterministic choice.

(tac_1 **THEN** tac_2) applies first tac_1, yielding a sequence of possible new states. Then it applies tac_2 to each possible state in the sequence and flattens the resulting sequence of sequences.

(tac_1 **ORELSE** tac_2) yields the result of tac_1 if non empty, and applies tac_2 otherwise.

(tac_1 **APPEND** tac_2) concatenates the results of tac_1 and tac_2.

It is important to understand the difference between **ORELSE** and **APPEND**. When using **ORELSE**, if the first branch of the choice is selected, Isabelle will never backtrack on the second one. Consequently,

$$(tac_1 \textbf{ ORELSE } tac_2) \textbf{ THEN } tac_3$$

is not equivalent to

$$(tac_1 \textbf{ THEN } tac_3) \textbf{ ORELSE } (tac_2 \textbf{ THEN } tac_3).$$

Indeed, if tac_1 yields a non empty result while (tac_1 **THEN** tac_3) does not, the application of the first expression will fail, while the application of the second one will amount to the application of (tac_2 **THEN** tac_3). In contrast, **THEN**

is right distributive on **APPEND**. Thus using **ORELSE** is more efficient than using **APPEND** but may result in an incomplete strategy.

When using a tactical such as **APPEND**, the proof search space may increase dramatically. Two tacticals that allow one to control the size of the search space are **DETERM** and **FILTER**.

(**DETERM** *tac*) makes *tac* deterministic: it returns only the head of the sequence resulting from the application of *tac*.

(**FILTER** *cond tac*) applies *tac* but returns only the states that satisfy the condition *cond*.

4 A general strategy

To write down a general tactic that is efficient enough for classical sequent-calculus, using Isabelle's basic tacticals and tactics, is easy. This is because there exist, for classical logic, sequent-calculi where all the rules are invertible. Consequently, each single resolution step may be performed deterministically. Unfortunately, this is not so in the case of linear logic.

In [2, 4], Galmiche and Perrier study the permutability of the inference rules of linear logic. The result of their work, for the multiplicative additive fragment, is summarised by the following table.

R_2 \ R_1	$\parr$	$\otimes$	&	$\oplus_L$	$\oplus_R$
$\parr$	*		*	*	*
$\otimes$	*	*	*	*	*
&	*		*		
$\oplus_L$	*	*	*	*	*
$\oplus_R$	*	*	*	*	*

A star, in this table, indicates that any scheme of the form

$$\dfrac{\dfrac{\vdots}{\dfrac{\cdot}{\cdot}\,R_1}}{\cdot}\,R_2$$

appearing in a proof may be replaced by the scheme:

$$\dfrac{\dfrac{\vdots}{\dfrac{\cdot}{\cdot}\,R_2}}{\cdot}\,R_1$$

Consequently, columns full of stars ($\parr$ and &) correspond to rules that can be pushed down the proof.

The above table suggests the general tactic specified by the following equation:

```
gen_tac i = axiom
            ORELSE
            (DETERM par) THEN gen_tac i
            ORELSE
            ((DETERM with) THEN gen_tac (i + 1)
                           THEN gen_tac i)
            ORELSE
            (((plus_L APPEND plus_R) THEN gen_tac i)
             APPEND
             ((times THEN gen_tac (i + 1))
                     THEN gen_tac i))
```

where the parameter i specifies the number of the subgoal to which the tactic is applied, and where "axiom", "par",... correspond to a single use of Isabelle's *resolve_tac* with the corresponding rule (except in the case of "times", where applications of the exchange rules are also needed in order to simulate associative, commutative unification).

The results of Galmiche and Perrier imply that *gen_tac* is a complete proof search strategy. Nevertheless, *gen_tac* is not satisfactory because it is inefficient. Consider the following provable sequent:

$$\vdash \underbrace{a \oplus \cdots \oplus a}_{n\times} \oplus b, \underbrace{a \oplus \cdots \oplus a}_{n\times} \oplus b^{\perp} \tag{5}$$

Proving this sequent with *gen_tac* gives the following timing (see Section 7 for hardware and software specifications):

$n =$	0	1	2	3	4	5
CPU time	0.05 sec	0.33 sec	1.84 sec	7.80 sec	29.88 sec	112.19 sec

These results are predictable because a simple analysis of *gen_tac* shows that the strategy is exponential. However, for a sequent as simple as (5), this inefficiency is unacceptable. Indeed, the additive fragment of linear logic is nc^1-complete and, consequently, polynomial algorithms exist [1].

The inefficiency of *gen_tac* is mainly due to the *or*-nodes of the proof search tree that are created when using the tactical **APPEND**. Nevertheless we may not use the more efficient tactical **ORELSE** everywhere without loosing completeness. Therefore we need criteria and heuristics that allow the *or*-nodes to be pruned but that also preserve completeness. In the remainder of this paper, we describe such criteria.

5 The multiplicative fragment

The multiplicative fragment is the one containing only the connectives $\otimes$ and $\parr$. The decision problem of this fragment is known to be NP-complete [7].

As we have seen, the $\parr$'s do not raise any difficulty: any formula $A \parr B$ appearing in a sequent to be proved may be replaced by the two formulas A and B. Therefore, the difficult case arises when all the formulas whose main connective is a $\parr$ have been simplified, i.e., when the sequent to be proved is of the following form:

$$\vdash (A_i \otimes B_i)_{i \in n}, (l_i)_{i \in m}, \tag{6}$$

where the l_i's are literals. In this case, we have to reduce Goal (6) to two new subgoals of the forms $\vdash A_k, \Gamma_1$ and $\vdash B_k, \Gamma_2$. There are two problems:

Problem 1: we do not know how to split the context $(A_i \otimes B_i)_{i \in n \setminus \{k\}}, (l_i)_{i \in m}$ into Γ_1 and Γ_2;
Problem 2: we do not know which $k \in n$ must be chosen.

Consequently, the (naive) resolution of Goal (6) gives rise to an *or*-node with $(n+1)\, 2^{n+m+1}$ branches, which is costly (even for a NP-complete problem).

Trying to overcome Problem 1

In order to attack Problem 1, we use a criterion that allows some of the branches to be eliminated. Such a criterion must obey two requirements:

- it must be based on a necessary condition of provability (i.e., equivalently, a sufficient condition of unprovability)
- it must be computable in polynomial time.

The first such condition that one may try is that, whenever a sequent is provable in the multiplicative fragment of linear logic, the number of positive occurrences of any literal in the sequent is equal to the number of its negative occurrences.

More precisely, define $\overline{\Gamma}$, the multiset of literals occuring in a sequent $\vdash \Gamma$ as follows:

$$\overline{(A_i)_{i \in n}} = \biguplus_{i \in n} \overline{A_i}$$

where, in the case of a formula,

i. $\overline{l} = \{l\}$,
ii. $\overline{A \parr B} = \overline{A} \uplus \overline{B}$,
iii. $\overline{A \otimes B} = \overline{A} \uplus \overline{B}$,

and where $\uplus$ denotes multiset union. Given a multiset of literals $\mathcal{A}$, define $\mathcal{A}^\perp$ to be the mutiset obtained by negating all the literals occuring in $\mathcal{A}$. Finally, call a multiset of literals $\mathcal{A}$ *perfectly balanced* if and only if $\mathcal{A} = \mathcal{A}^\perp$.

The following holds:

Proposition 5.1 *If a sequent $\vdash \Gamma$ is provable in the $\otimes\parr$-fragment of linear logic then $\overline{\Gamma}$ is perfectly balanced.*

Proof. An obvious induction on the derivation of $\vdash \Gamma$. □

Tammet mentions the above criterion in [10] and claims that using a rudimentary form of it improved the efficiency of his prover by a factor of 3 to 10. Independently we implemented the above criterion and noticed a significant improvement (a factor of 10) in efficiency. However we also discovered that the criterion itself could be improved.

Let us introduce our improved criterion by an example. Consider the following goal:

$$\vdash b \otimes c^{\perp}, a^{\perp} \otimes a, b^{\perp}, c \tag{7}$$

A priori, there are sixteen ways of applying a $\otimes$-resolution step to Goal (7). Fourteen of these sixteen ways will be rejected by the criterion of Proposition 5.1. The two remaining possibilities would yield the two following pairs of new subgoals:

$$\vdash b, a^{\perp} \otimes a, b^{\perp} \qquad\qquad \vdash c^{\perp}, c \tag{8}$$

$$\vdash b, b^{\perp} \qquad\qquad \vdash c^{\perp}, a^{\perp} \otimes a, c \tag{9}$$

Then, a further application of a $\otimes$-resolution step, together with the criterion of Proposition 5.1, would allow both the first subgoal of (8) and the second subgoal of (9) to be found unprovable. Nonetheless, the reason why these two subgoals are eventually rejected may be already observed in Goal (7): the only negative occurrence of the literal a appears on the lefthand side of a tensor while the corresponding positive occurrence appears on the righthand side of the same tensor. Therefore, a criterion designed to detect such *pathologies* could be used to reject Goal (7) *a priori*.

Our modified criterion, which generalises the above observation, is based on the next proposition:

Proposition 5.2 *If a sequent* $\vdash A \otimes B, \Gamma$ *is provable in the* $\otimes\parr$*-fragment of linear logic then*

1. $\overline{A}^{\perp} \subset \overline{\Gamma}$ *(and, consequently, the multiset* $\overline{\Gamma} \setminus \overline{A}^{\perp}$ *is defined),*
2. *symmetrically,* $\overline{B}^{\perp} \subset \overline{\Gamma}$*,*
3. $\overline{A} \uplus (\overline{\Gamma} \setminus \overline{B}^{\perp})$ *is perfectly balanced,*
4. *symmetrically,* $\overline{B} \uplus (\overline{\Gamma} \setminus \overline{A}^{\perp})$ *is perfectly balanced.*

Proof. By induction on the derivation of the sequent. □

By using a criterion based on Proposition 5.2 instead of the one based on Proposition 5.1, we improved further the efficiency of our proof search strategy (again by a factor about 10).

Trying to overcome Problem 2

The above criterion is of no use in pruning the *or*-branching due to Problem 2. In order to tackle this second problem, we use a heuristic based on Girard's sequentialisation theorem [5, 6].

It is rather easy to prove that, whenever a sequent $\vdash (A_i)_{i\in n}$ is provable, there exists a proof net whose conclusions are the formulas A_i's. The converse, which is more difficult to establish, is given by Girard's sequentialisation theorem: any proof net whose conclusions are the formulas $(A_i)_{i\in n}$ may be transformed into a sequential proof of the sequent $\vdash (A_i)_{i\in n}$ (in polynomial time).

Constructing a proof net for a given sequent is as complex as searching a sequential proof of the sequent. Nevertheless, gross approximations of proof nets may be constructed easily.

Let $\Gamma = (A_i)_{i\in n}$ be a sequence of formulas. We define $\Pi(\Gamma)$, the *overlinked proof structure* associated to Γ as the graph made up of the syntactic trees of each formula A_i together with additional edges between each occurence of a literal l and all the occurences of its negation $l^\perp$. Notice that any proof net is a subgraph of some overlinked proof structure (see Fig. 1 and 2).

Consider an overlinked proof structure Π. We say that a vertex $(A \otimes B)$ is a *splitting tensor* of Π if and only if

1. $(A \otimes B)$ is a conclusion of Π: i.e., the vertex $(A \otimes B)$ has only two adjacent edges: the ones joining respectively A and B to $(A \otimes B)$,
2. the removal of the vertex $(A \otimes B)$, together with the two adjacent edges, disconnects Π (in the graph-theoretic sense).

Our heuristic, which takes advantage of the notion of splitting tensor, is based on the next proposition.

Proposition 5.3 *Consider a sequent $\vdash A\otimes B, \Gamma$ and let $(A\otimes B)$ be a splitting tensor of $\Pi(A \otimes B, \Gamma)$. If $\vdash A \otimes B, \Gamma$ is provable, there exist Γ_1 and Γ_2 such that:*

1. $\Gamma_1 \uplus \Gamma_2 = \Gamma$,
2. $\vdash A, \Gamma_1$ *and* $\vdash B, \Gamma_2$ *are both provable.*

Proof. $\vdash A \otimes B, \Gamma$ being provable, there exists a proof net π whose conclusions are $A \otimes B, \Gamma$. By definition of overlinked proof structure, $\pi \subset \Pi(A \otimes B, \Gamma)$. Therefore the removal of the splitting tensor $A \otimes B$ splits π into two distinct proof nets π_A and π_B whose conclusions contains respectively A and B. Let us call Γ_1 (respectively, Γ_2) the other conclusions of π_A (respectively, π_B). By the sequentialisation theorem, $\vdash A, \Gamma_1$ and $\vdash B, \Gamma_2$ are both provable. □

Our proof search strategy is based on Proposition 5.3 as follows: each time we are facing a sequent akin to Goal 6 we first compute the corresponding overlinked proof structure and try to find a splitting tensor. Note that when we succeed in finding a splitting tensor, we also know how to split the context Γ into

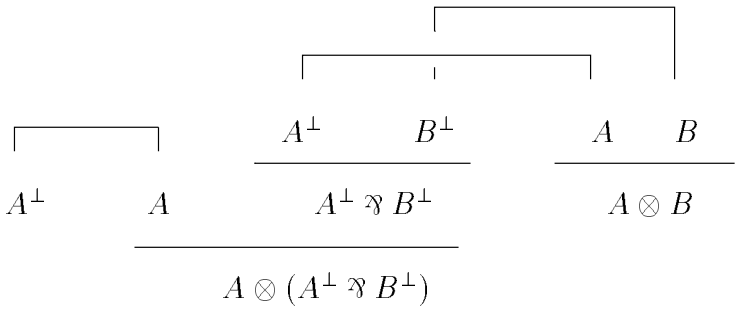

Fig. 1. A proof net

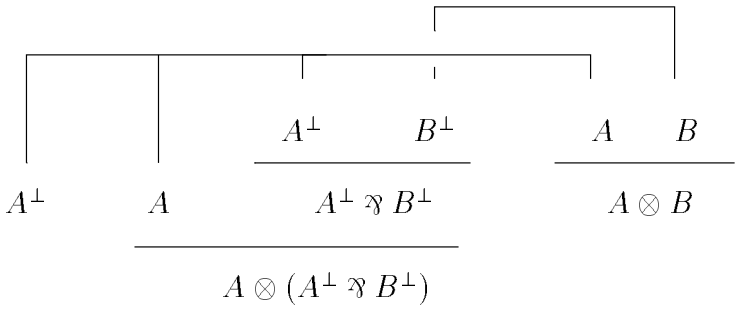

Fig. 2. The corresponding overlinked proof structure

Γ_1 and Γ_2. Indeed, by removing the splitting tensor $A \otimes B$ from $\Pi(A \otimes B, \Gamma)$, we split $\Pi(A \otimes B, \Gamma)$ into two graphs Π_A and Π_B such that $\pi_A \subset \Pi_A$ and $\pi_B \subset \Pi_B$. Moreover, the conclusions of π_A (respectively π_B) are the conclusions of Π_A (respectively Π_B). Therefore, when the heuristics succeeds, the resolution of Goal (6) becomes deterministic: the $(n+1)\,2^{n+m+1}$ possibilities are replaced by one.

The implementation of the above heuristic improved dramatically the efficiency of our proof search strategy. The experiments that we conducted indicate that the quality of the results presented in Section 7 is due mainly to the heuristic based on Proposition 5.3.

6 The multiplicative additive fragment

By adding $\oplus$ and $\&$ to the multiplicative fragment, we obtain the multiplicative additive fragment of linear logic, which is known to be PSPACE-complete [7].

As we have seen in Section 4, the $\parr$'s and the $\&$'s are not problematic: we may start our proof search by resolving deterministically all the formulas whose main connective is $\parr$ or $\&$. The difficult case consists in resolving a sequent of the following form:

$$\vdash (A_i \otimes B_i)_{i \in n}, (C_i \oplus D_i)_{i \in m}, (l_i)_{i \in l}. \tag{10}$$

For such a sequent, the problems to solve are the following:

Problem 1: we do not know how to split the context when resolving a formula $A_i \otimes B_i$,
Problem 2: we do not know how to chose $i \in n$ when resolving such a formula,
Problem 3: we do not know if we must use the right or the left introduction rule to resolve a formula $C_i \oplus D_i$,
Problem 4: we do not know how to chose $i \in m$ when resolving such a formula,
Problem 5: we do not know whether we may resolve $\oplus$ before $\otimes$ (or the converse).

Consequently, the resolution of Goal (10) gives rise to a *or*-node with $(n+1)\,2^{n+m+l+2} + 2\,(m+1)$ branches.

The solution to these problems is similar to the one used in the previous section: we develop criteria for pruning the *or*-nodes.

The heuristic based on Proposition 5.3 is still valid in the multiplicative additive fragment, when facing a sequent akin to Goal 10. The proof, however, is much more complicate to state because proof-nets in this fragment may contain boxes. Nevertheless the argument is the same because, on the one hand, the boxes of a multiplicative additive proof-net may be replaced by n-ary axiomatic links and, on the other hand, the relation of interest in an overlinked proof structure is not the adjacency relation but its transitive closure restricted to the conclusions of the overlinked proof structure.

In contrast, Proposition 5.2 does not hold in the presence of the additives for two reasons:

1. the backward chaining of the introduction rules for $\oplus$ allows literals to disapear,
2. the backward chaining of the introduction rule for & allows literals to be multiplied.

In order to adapt Proposition 5.2, we introduce the notion of forced literal (because of 1.) and we work with sets rather than multisets (because of 2.).

Define $\overline{\Gamma}$ as the set of literals occuring in a sequent $\vdash \Gamma$. Define also $\overline{\overline{\Gamma}}$, the set of *forced literals* occuring in a sequent $\vdash \Gamma$, as follows:

$$\overline{\overline{(A_i)_{i\in n}}} = \bigcup_{i\in n} \overline{\overline{A_i}}$$

where, in the case of a formula,

i. $\overline{\overline{l}} = \{l\}$,
ii. $\overline{\overline{A \parr B}} = \overline{\overline{A}} \cup \overline{\overline{B}}$,
iii. $\overline{\overline{A \otimes B}} = \overline{\overline{A}} \cup \overline{\overline{B}}$,
iv. $\overline{\overline{A \,\&\, B}} = \overline{\overline{A}} \cup \overline{\overline{B}}$,

v. $\overline{\overline{\overline{A \oplus B}}} = \emptyset$.

Given a set of literals $\mathcal{A}$, define $\mathcal{A}^\perp$ to be the set obtained by negating the literals occuring in $\mathcal{A}$.

Proposition 5.2 may be adapted as follows.

Proposition 6.1 *If a sequent* $\vdash A \otimes B, \Gamma$ *is provable in the* $\otimes \parr \oplus \&$*-fragment of linear logic then*

1. $(\overline{\overline{\Gamma, A, B}})^\perp \subset \overline{\Gamma, A, B}$,
2. $\overline{\overline{A}}^\perp \subset \overline{\Gamma, A}$,
3. *symmetrically,* $\overline{\overline{B}}^\perp \subset \overline{\Gamma, B}$.

Proof. By induction on the derivation of the sequent. □

The new pruning criterion based on Proposition 6.1 is of course valid for the multiplicative fragment but is less sharp than the criterion based on Proposition 5.2. For instance, the sequent $\vdash A \otimes B, A^\perp, B, B^\perp$, which is rejected by criterion 5.2, is accepted by criterion 6.1.

In practice, to replace criterion 5.2 by criterion 6.1, in the multiplicative case decreases considerably the efficiency of the proof search strategy. This observation suggests another heuristic. When trying to prove multiplicative additive goals, one may generate subgoals that are purely multiplicative and for which one may specialise the proof search strategy.

Our general proof search strategy generalises this last heuristic: each time we attack a subgoal we check the fragment to which it belongs and, accordingly, we specialise the general strategy as follows.

$\otimes \parr \oplus$-fragment According to the table given in Section 4, the only rule that possibly prevents Rules (Plus-L) and (Plus-R) to be pushed down in a proof is Rule (With). Therefore, in the fragment without &, Problems 4 and 5 vanish: the $\oplus$'s may be resolved deterministically before the $\otimes$'s (nevertheless, the undeterminism due to Problem 3 remains).

$\otimes \parr$-fragment This is the multiplicative fragment for which we may use the criterion of Proposition 5.2.

$\otimes \oplus$-fragment In this fragment, Problems 4 and 5 vanish because of the absence of &'s. Moreover, Rules (Times) can also be pushed down because the fragment does not contain $\parr$'s either. Therefore, the $\oplus$'s may be resolved deterministically before the $\otimes$'s. Then, the $\otimes$'s may be resolved deterministically as well.

$\& \oplus$-fragment This is the additive fragment for wich the decision problem is polynomial hard. In this fragment, provable sequents are made of exactly two formulas.

7 Implementation and experimental results

The different strategies described in this paper have been implemented in Isabelle. The code is about 500 lines of standard ML. Experiments were conducted on a SPARC station IPX, using the New-Jersey standard ML compiler. Among others, we have experimented with the formulas LMSS-2, LMSS-3, and LMSS-4 given in [10]. These formulas, which were introduced in [7] to encode Boolean circuits, are as follows:

LMSS-2 (provable)

$$
\begin{array}{l}
\vdash\ x^{\perp} \parr \\
\quad y^{\perp} \parr \\
\quad g \parr \\
\quad ((c \otimes h \otimes g^{\perp}) \oplus (c^{\perp} \otimes h^{\perp} \otimes g) \oplus (c \otimes h^{\perp} \otimes g) \oplus (c^{\perp} \otimes h \otimes g)) \parr \\
\quad ((x \otimes (x^{\perp} \parr x^{\perp})) \oplus (x^{\perp} \otimes (x \parr x))) \parr \\
\quad ((y \otimes (y^{\perp} \parr y^{\perp})) \oplus (y^{\perp} \otimes (y \parr y))) \parr \\
\quad ((b \otimes c) \oplus (b^{\perp} \otimes c^{\perp})) \parr \\
\quad ((a \otimes y \otimes b^{\perp}) \oplus (a^{\perp} \otimes y^{\perp} \otimes b) \oplus (a \otimes y^{\perp} \otimes b) \oplus (a^{\perp} \otimes y \otimes b)) \parr \\
\quad ((x \otimes a) \oplus (x^{\perp} \otimes a^{\perp})) \parr \\
\quad ((e \otimes h) \oplus (e^{\perp} \otimes h^{\perp})) \parr \\
\quad ((d \otimes x \otimes e^{\perp}) \oplus (d^{\perp} \otimes x^{\perp} \otimes e) \oplus (d \otimes x^{\perp} \otimes e) \oplus (d^{\perp} \otimes x \otimes e)) \parr \\
\quad ((y \otimes d) \oplus (y^{\perp} \otimes d^{\perp}))
\end{array}
$$

LMSS-3 (provable)

$$
\begin{array}{l}
\vdash\ s \parr \\
\quad (s^{\perp} \otimes ((r \parr x) \,\&\, (r \parr x^{\perp}))) \parr \\
\quad (r^{\perp} \otimes ((q \parr y) \oplus (q \parr y^{\perp}))) \parr \\
\quad g \parr \\
\quad (q^{\perp} \otimes (((c \otimes h \otimes g^{\perp}) \oplus (c^{\perp} \otimes h^{\perp} \otimes g) \oplus (c \otimes h^{\perp} \otimes g) \oplus (c^{\perp} \otimes h \otimes g)) \parr \\
\qquad ((x \otimes (x^{\perp} \parr x^{\perp})) \oplus (x^{\perp} \otimes (x \parr x))) \parr \\
\qquad ((y \otimes (y^{\perp} \parr y^{\perp})) \oplus (y^{\perp} \otimes (y \parr y))) \parr \\
\qquad ((b \otimes c) \oplus (b^{\perp} \otimes c^{\perp})) \parr \\
\qquad ((a \otimes y \otimes b^{\perp}) \oplus (a^{\perp} \otimes y^{\perp} \otimes b) \oplus (a \otimes y^{\perp} \otimes b) \oplus (a^{\perp} \otimes y \otimes b)) \parr \\
\qquad ((x \otimes a) \oplus (x^{\perp} \otimes a^{\perp})) \parr \\
\qquad ((e \otimes h) \oplus (e^{\perp} \otimes h^{\perp})) \parr \\
\qquad ((d \otimes x \otimes e^{\perp}) \oplus (d^{\perp} \otimes x^{\perp} \otimes e) \oplus (d \otimes x^{\perp} \otimes e) \oplus (d^{\perp} \otimes x \otimes e)) \parr \\
\qquad ((y \otimes d) \oplus (y^{\perp} \otimes d^{\perp}))))
\end{array}
$$

LMSS-4 (unprovable)

$$
\begin{array}{l}
\vdash\ s \parr \\
\quad (s^{\perp} \otimes ((r \parr x) \oplus (r \parr x^{\perp}))) \parr \\
\quad (r^{\perp} \otimes ((q \parr y) \parr (q \parr y^{\perp}))) \parr \\
\quad g \parr \\
\quad (q^{\perp} \otimes (((c \otimes h \otimes g^{\perp}) \oplus (c^{\perp} \otimes h^{\perp} \otimes g) \oplus (c \otimes h^{\perp} \otimes g) \oplus (c^{\perp} \otimes h \otimes g)) \parr \\
\qquad ((x \otimes (x^{\perp} \parr x^{\perp})) \oplus (x^{\perp} \otimes (x \parr x))) \parr \\
\qquad ((y \otimes (y^{\perp} \parr y^{\perp})) \oplus (y^{\perp} \otimes (y \parr y))) \parr \\
\qquad ((b \otimes c) \oplus (b^{\perp} \otimes c^{\perp})) \parr \\
\qquad ((a \otimes y \otimes b^{\perp}) \oplus (a^{\perp} \otimes y^{\perp} \otimes b) \oplus (a \otimes y^{\perp} \otimes b) \oplus (a^{\perp} \otimes y \otimes b)) \parr \\
\qquad ((x \otimes a) \oplus (x^{\perp} \otimes a^{\perp})) \parr \\
\qquad ((e \otimes h) \oplus (e^{\perp} \otimes h^{\perp})) \parr \\
\qquad ((d \otimes x \otimes e^{\perp}) \oplus (d^{\perp} \otimes x^{\perp} \otimes e) \oplus (d \otimes x^{\perp} \otimes e) \oplus (d^{\perp} \otimes x \otimes e)) \parr \\
\qquad ((y \otimes d) \oplus (y^{\perp} \otimes d^{\perp}))))
\end{array}
$$

Tammet's bottom-up prover does not succeed in proving these three formulas (after a 14-hour search, the prover had generated up to 500 million of subgoals [10]). In contrast, our Isabelle implementation gives the following timing:

LMSS-2	LMSS-3	LMSS-4
43 sec	5 min 55 sec	3 min 35 sec

Tammet has also implemented a resolution prover based on the top-down strategies that he discusses in [10]. This resolution prover proves LMSS-2 and LMSS-3 in about 1 min, and takes about 3 min to reject LMSS-4 as unprovable.

Tammet's results may seem to indicate that forward proof search in linear logic is much more efficient than backward chaining. In contrast, our results demonstrate that backward proof search is feasible when appropriate criterion for pruning the proof search tree are used. Our results concern only the constant-free, multiplicative, addive fragment of linear logic. They are encouraging. We must now generalise our work to full linear logic: i.e. to allow for constants, exponentials, and quantifiers.

References

[1] P. Bradford, J.-Y. Marion, and L. Moss. The additive fragment of linear logic is nc^1-complete. In D. Leivant, editor, *Logic complexity and computation*, page 30, October 1994.

[2] G. Perrier D. Galmiche. Automated deduction in additive and multiplicative linear logic. In A. Nerode and M. Taitslin, editors, *Proceedings of the Second International Symposium on Logical Foundations of Computer Science – Tver'92*, pages 151–162. Lecture Notes in Computer Science, 620, Springer Verlag, 1992.

[3] Jean Gallier. Constructive logic part II: Linear logic and proof nets. Technical Report 9, Digital Equipment Corporation, Paris, 1991.

[4] D. Galmiche and G. Perrier. On proof normalisation in linear logic. *Theoretical Computer Science*, 1994. to appear in december 1994.
[5] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
[6] J.-Y. Girard. Quantifiers in linear logic II. Technical Report 19, Equipe de Logique Mathématique, Université de Paris VII, 1991.
[7] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure ans Applied Logic*, 56:239–311, 1992.
[8] L. C. Paulson. *Isabelle, a generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
[9] L.C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.
[10] T. Tammet. Proof search strategies in linear logic. Technical Report 70, Department of Computer Science, Chalmers University of Technology, Göteborg, Sweden, 1993.
[11] Anne Troelstra. *Lectures on Linear Logic*, volume 29 of *CSLI Lecture Notes*. Center for the Study of Language and Information, 1992.