A procedure for automatic proof nets construction

Didier Galmiche, Guy Perrier

A procedure for automatic proof nets construction

Didier GALMICHE and Guy PERRIER

CRIN - CNRS - INRIA Lorraine
Campus Scientifique - B.P. 239
54506 Vandœuvre-les-Nancy Cedex
France
e-mail: galmiche{perrier}@loria.crin.fr

Abstract

In this paper, we consider multiplicative linear logic (MLL) from an automated deduction point of view. Linear logic is more expressive than classical and intuitionistic logic and has an undirectional character due to the particular treatment of negation and the absence of structural rules. Considering this new logical framework to make logic programming or programming with proofs (extracting programs from proofs), a better comprehension of proofs in MLL (proof nets) is necessary and automated deduction has to be studied. Knowing that the multiplicative part of linear logic is decidable, we propose a new algorithm to construct automatically a proof net for a given sequent in MLL with proofs of termination, correctness and completeness. It can be considered as a more direct and implementation oriented way to consider automated deduction in linear logic.

1 Introduction

Computer scientists can consider logic with two points of view: it can be an external reference to which they report during their activity, for example, to make correctness proofs of programs; it can be integrated as an internal tool for programming. In this work, we consider the second point of view. Intuitionism [17] has given at first a logical tool to approach programming in this connection. Centered around a constructive vision of truth, it allows to consider proofs as functions, lambda-calculus being a good framework to code them as programs. Considering the *Curry-Howard* correspondence [12] for which a proof is a λ-term or a program and a formula is a type or a specification, the automatic program synthesis can be viewed as issued from automatic deduction [7]: for a given specification, we construct a proof to extract a program from it. Taking into account the unsymmetry of intuitionism that distinguishes hypotheses and conclusion it can be difficult to consider concurrency and reactive systems in this framework.
The linear logic created by J.-Y. Girard [10, 11] appears as a potential good logical framework to consider concurrency. Its principal characteristic is to be a logic for actions introducing notions like controlled and strict management and resource consumption. It conserves a constructive character without having the default of absence of symmetry. If we want to consider the *proofs as programs* approach in linear logic we have to abord it from the (automatic) deduction point of view. To consider linear logic from a computational interpretation (of the logic) can lead to a process paradigm opening up a new approach of parallel implementation of functional languages and of typed concurrent programming [1]. The point is to better understand what a proof is in linear logic consisting in a particular graph named *proof net* [6, 10].

But before considering the computational aspect and the synthesis of programs from proof nets, for example using an adequate refinement of typed λ-calculus [16], one should be able to mechanize the construction of proofs in linear logic. We know that propositional linear logic is undecidable [14] but it is not the case for multiplicative part of it. In this paper, we give an algorithm to mechanize the construction of a proof net for a given sequent in multiplicative linear logic (MLL) sequent calculus. Moreover proofs of correction, correctness and completeness are given. From this point it will be able to study from an automated deduction point of view the application of deduction in linear logic in different fields as programming logic [2, 13] or plans generation [15]. In section 2, we give a concise presentation of linear logic and in section 3 we make precise the concepts of *logical structure* and of *proof net*. In section 4, we focus on multiplicative linear logic where the proof net notion is clearly defined and present the principle of automatic construction of proof net. In section 5, before considering the algorithm, we illustrate the principle by an example. Section 6 contains the complete presentation of the algorithm for the automatic proof net construction with the proofs of termination, correctness and completeness. Section 7 presents some conclusions.

2 Linear logic

Linear logic (LL) has been introduced recently by Girard [10] as a logic of actions. Born from the semantics of second order lambda-calculus, linear logic is more expressive than traditional logic (classical or intuitionistic ones). Characterized by the absence of structural rules and of a specific treatment of the negation, linear logic has proofs that can be considered as actions and introduces a dynamical resource management in these proofs without directional character (no distinction between input and output).

2.1 Sequent calculus for LL

The deduction, in the linear meaning, is viewed as an interaction between hypotheses and conclusions. We list the rules of inference of the calculus of sequents for linear logic. Using a presentation with sequents without left-hand side, we have:

Identity rule

$$\frac{}{\vdash A, A^{\perp}}\ Id \qquad \frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\ Cut$$

Structural rules

$$\frac{\vdash \Gamma}{\vdash \Gamma'}\ Ex \quad \text{where } \Gamma' \text{ is the result of a permutation of } \Gamma.$$

Logical rules

Negation

$A^{\perp\perp} = A \qquad (A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp} \qquad (A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp}$
$(A \& B)^{\perp} = A^{\perp} \oplus B^{\perp} \qquad (A \oplus B)^{\perp} = A^{\perp} \& B^{\perp}$
$(!A)^{\perp} = ?A^{\perp} \qquad (?A)^{\perp} = !A^{\perp}$
$(\forall x A)^{\perp} = \exists x A^{\perp} \qquad (\exists x A)^{\perp} = \forall x A^{\perp}$

Multiplicative operators

$$\frac{\vdash \Gamma_1, A_1 \quad \vdash \Gamma_2, A_2}{\vdash \Gamma_1, \Gamma_2, A_1 \otimes A_2}\ \otimes \qquad \frac{\vdash \Gamma, A_1, A_2}{\vdash \Gamma, A_1 \wp A_2}\ \wp$$

Additive operators

$$\frac{\vdash \Gamma, A_1 \quad \vdash \Gamma, A_2}{\vdash \Gamma, A_1 \& A_2}\ \& \qquad \frac{\vdash \Gamma, A_1}{\vdash \Gamma, A_1 \oplus A_2}\ \oplus_1 \qquad \frac{\vdash \Gamma, A_2}{\vdash \Gamma, A_1 \oplus A_2}\ \oplus_2$$

Exponential operators

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\ ! \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ d? \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ w? \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ c?$$

Quantifiers

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x A}\ \forall \quad \textit{where } x \textit{ is not free in } \Gamma \qquad \frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x A}\ \exists$$

The logic nature is determined by the structural rules. In linear logic, the *weakening* rule is rejected to establish a linear dependence between hypothesis and conclusion. Moreover, the *contraction* rule is rejected to establish a strict and explicit resources management. The *exchange* rule is conserved but it is possible to consider a linear logic without this rule that would be called a non commutative linear logic. Then linear logic is a refinement of the classical one. To be complete, let us precise that the rejected structural rules are reintroduced through modal operators (?, !) allowing a local and controlled use.
Linear negation can be introduced by

$$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^{\perp}, \Delta} \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma, B^{\perp} \vdash \Delta}\ .$$

giving a fundamental symmetry that can transform each sequent into a sequent without left hand side: one does not distinguish hypothesis and conclusion. For example: $A, B, C^{\perp} \vdash D, E^{\perp}$ can be written $\vdash A^{\perp}, B^{\perp}, C^{\perp\perp}, D, E^{\perp}$.
The rejection of weakening leads to the dissociation of the classical *and* and *or* :
- *and* is translated by $\otimes$ (*times*) and & (*with*), both connectors expressing that the conditions A_1 and A_2 are realizable, both, at the same time, with $\otimes$, and only one, at once, with &.
- *or* is translated by $\wp$ (*par*) and $\oplus$ (*plus*). The $\wp$ connector expresses a dependance between the two conditions A_1 et A_2: if A_1 is not realized then A_2 will be and if A_2 is not realized A_1 will be. The connector $\oplus$ expresses the choice of one condition A_1 or A_2, these being completely independent.
$\wp$ is the dual of $\otimes$ and $\oplus$ is the dual of &. In fact, $(A \otimes B)^{\perp} \equiv A^{\perp} \wp B^{\perp}$ and $(A \& B)^{\perp} \equiv A^{\perp} \oplus B^{\perp}$.
times and *par* do not modify the resources range but only the structure. That is why the operators *times* and *par* have **intensional** or **multiplicative** character.
with and *plus* modify the resources range. That is why one says that *with* and *plus* have an **extensional** or **additive** character.
Moreover the cut rule has a dynamical character, allowing the construction of complex actions from elementary ones. This creation power of the rule is linked to the Gentzen Haupsatz [9], proved for linear logic by Girard [10]. The *linear logic* introduces a *dynamic resources management* in the proofs. Moreover, we have a *symmetry* : involution of negation, no distinction between hypotheses and conclusion. We can associate different semantics to linear logic: for

example, a phase semantics and a coherent space semantics. A broad explanation for the meaning and the purpose of linear logic is given in [10]. Here we only consider the multiplicative part of linear logic (MLL) from the deduction theory point of view.

3 The proof nets

This concept has been created by J.Y. Girard [10] to precise the particular nature of proofs in linear logic. We give here a precise and little different (compared to the classical one) presentation of this concept. It corresponds in multiplicative linear logic to the double worry of machine implementation and study of information circulation in such structure. Here the nodes correspond to the information processors and the edges to channels for information circulation.

3.1 From sequential proofs to proof nets

With the rules of linear logic, we can develop proofs in linear logic. For example:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}\,Id \quad \dfrac{}{\vdash B, B^{\perp}}\,Id}{\vdash A \otimes B, A^{\perp}, B^{\perp}}\otimes \quad \dfrac{}{\vdash C, C^{\perp}}\,Id}{\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp}, C}\otimes}{\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp} \wp C}\wp$$

is a proof (Π_1) of the sequent $\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp} \wp\, C$.
But it is not the only one; here is another one (Π_2):.

$$\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}\,Id \quad \dfrac{\dfrac{}{\vdash B, B^{\perp}}\,Id \quad \dfrac{}{\vdash C, C^{\perp}}\,Id}{\vdash B^{\perp} \otimes C^{\perp}, B, C}\otimes}{\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp}, C}\otimes}{\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp} \wp C}\wp$$

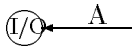
These two proofs differ only by the order in which rules are used which is not essential.
The idea is to represent a proof by a graph called *proof net*: A sequent $\vdash A_1, ..., A_n$ is represented by a *box* with n inputs-outputs labelled respectively $A_1, ..., A_n$. The internal structure of this box will represent a proof of the sequent and will constitute the *proof net* .
A sequent being an *axiom* has, by definition, no proof associated. It will be represented by a black box only defined by its interface.
Let us define the proof nets by precising at first their syntactic form named logical structure.

3.1.1 Logical structure definition

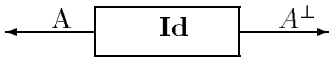
The edges are oriented and labelled by formulas of multiplicative linear logic. We have five node types and the node type determines the number of edges to which it is connected and the particular relationship linking formulas attached to these edges.

Node types of a logical structure

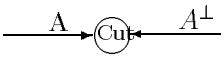
Input-output



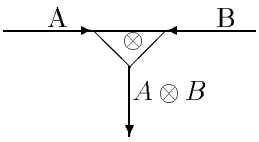
Identity



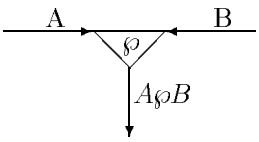
Cut



Times



Par



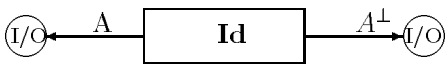
A graph respecting this syntactic form is a *logical structure*. Of course, a given logical structure does not represent necessarily a proof in linear logic. It has to verify certain criteria to become a *proof net*. We will define it inductively from elementary proof nets associated with identity axioms. To each inference rule of linear logic corresponds a rule for the proof net construction.

3.1.2 Proof nets construction rules

Identity

To identity rule $\dfrac{}{\vdash A, A^\perp}$ corresponds the construction rule:

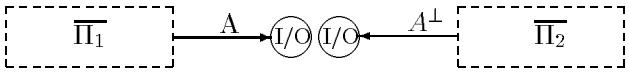
The logical structure below is an elementary proof net



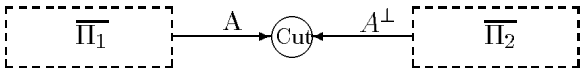
Cut

To cut rule: $\dfrac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$ corresponds the construction rule:

If $\overline{\Pi_1}$ *and* $\overline{\Pi_2}$ *are proof nets of the form:*



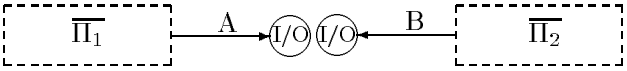
then the following logical structure is a proof net:



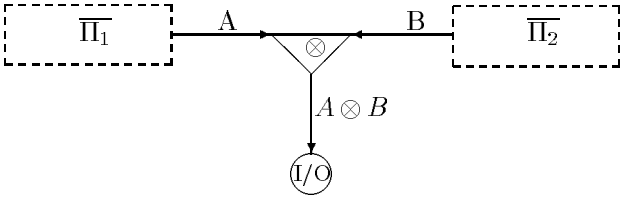
Times

To *times* rule: $\dfrac{\vdash \Gamma_1, A_1 \quad \vdash \Gamma_2, A_2}{\vdash \Gamma_1, \Gamma_2, A_1 \otimes A_2}$ corresponds the construction rule:

If $\overline{\Pi_1}$ *and* $\overline{\Pi_2}$ *are proof nets:*



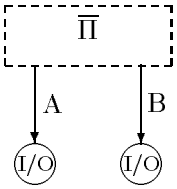
then the following logical structure is a proof net:



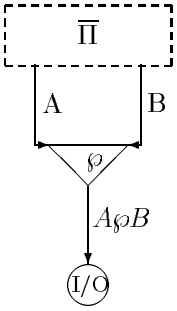
Par

To the *par* rule: $\dfrac{\vdash \Gamma, A_1, A_2}{\vdash \Gamma, A_1 \wp A_2}$ corresponds the construction rule:

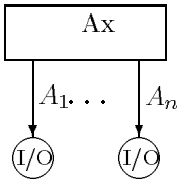
If $\overline{\Pi}$ *is a proof net of the form below:*



then the following logical structure is a proof net:



Remark 3.1 *If one considers a particular theory defined by a set of proper axioms, one associates to each axiom* Ax *of the form* $\vdash A_1, A_2, ..., A_n$ *an elementary proof net of the form:*



Remark 3.2 *We can extend the construction rules for the treatment of the additives connectives but we consider here only classical proof nets in MLL.*

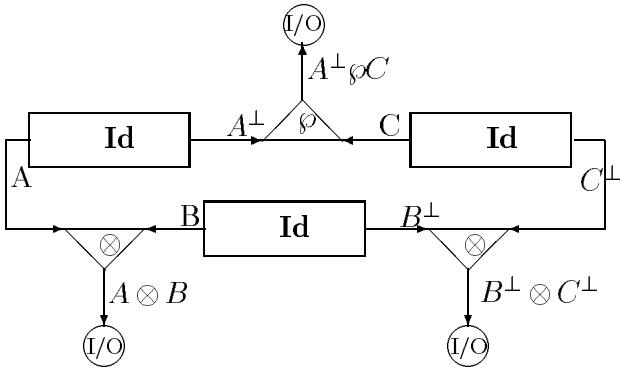
3.1.3 Correspondence between sequential proofs and proof nets

The rules defined above allow to associate naturally a proof net $\overline{\Pi}$ to a sequential proof Π in multiplicative linear logic. Hence the following theorem:

Theorem 3.1 *(Girard)*
A unique proof net $\overline{\Pi}$ *corresponds to a sequential proof* Π *in multiplicative linear logic. The application:* $\Pi \longrightarrow \overline{\Pi}$ *is surjective but not injective.*

3.1.4 Example

Let us illustrate how one can construct at the same time the sequential proof Π_1 and the proof net $\overline{\Pi_1}$. We want to construct the proof net corresponding to the proof of the sequent:
$\vdash A \otimes B, B^{\perp} \otimes C^{\perp}, A^{\perp} \wp C$
The result is the following proof net:



Whatever order is used to apply the construction rules, the final proof net is the same. The set of input/output with the opening on edges constitutes the external interface of the box associated to the sequent. This presentation of proof nets is adapted to our problem. Now having presented the logical framework and the concept of proof net, we can begin to give the general basic principle of our algorithm to automate the proof net construction from a given sequent (in multiplicative linear logic).

4 Principle of automatic proof nets construction

Usually proofs, in a given logic, can be constructed in two different ways: either by forward chaining, guided by hypotheses, or by backward chaining, guided by the conclusion.

We can consider these two methods for proof nets construction. The first one considering elementary proof nets, associated with identity axioms and using construction rules: it is a step by step construction through successive connections, resulting in more complex proof nets. The second one, considering the final proof net as a goal characterized by the inputs and ouputs, we split it into subgoals until we obtain elementary proof nets. The latter approach is the subject of other works not mentioned in this paper and we only consider the former one.
Let us begin to explain our problem. It consists in having an algorithm allowing to decide if a given sequent $\vdash A_1, A_2, ..., A_n$ in multiplicative linear logic is provable and if it is the case, to construct an associated proof net. According to the Gentzen Haupsatz, we can only consider proofs without cuts.

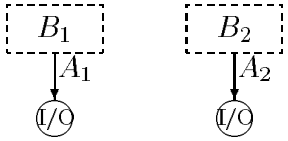
4.1 Terminal branches

The formulae $A_1, A_2, ..., A_n$ of the sequent to prove determine what we call the terminal branches of the net to construct.

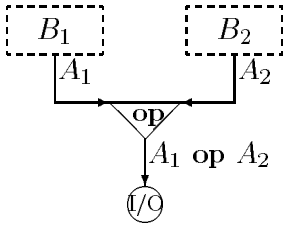
Definition 4.1 *A terminal branch is defined by induction by the three following rules.*

1. *If A is an atom (positive or negative)) then the following graph is a terminal branch.*

2. *If B_1 and B_2 are terminal branches (as indicated in the figure)*



then the following graph is a terminal branch:



where **op** *is $\otimes$ or $\wp$.*

Considering this definition, we can associate a terminal branch B_i with each formula A_i of the sequent to prove.

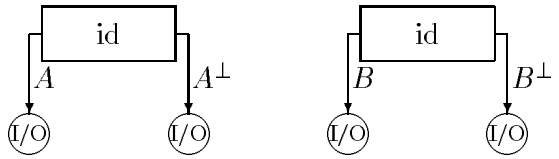
4.2 Free edges association by duality

After the creation of the terminal branches $B_1, B_2, ..., B_n$, we have to associate their free edges by duality.

Definition 4.2 *i) An edge is free, in a terminal branch, if it has no start node.*
ii) Two edges are dual if they are labelled by two formulae that are negation of the other ones.

If we are not careful, an association by duality of two free edges can lead to several disjoint proof nets. For example, the sequent $\vdash A, A^{\perp}, B, B^{\perp}$ is not provable. But what about a proof net construction attempt from it ? It begins by considering 4 terminal branches

An association by duality of free edges can lead to the following result:



To avoid such a schema, we distinguish one the terminal branches chosen as start point of the proof net in construction $\mathcal{G}$.

Definition 4.3 *A proof net in construction $\mathcal{G}$ is a connected graph obtained by association of dual free edges of terminal branches, using identity axioms.*

Then, we associate only from free edges of $\mathcal{G}$ so as to preserve the connexity. The construction stops when $\mathcal{G}$ has no more free edges. There are two possibilities: 1) terminal branches remain that are not connected to $\mathcal{G}$ (it is not the expected result). 2) No terminal branches remain that is the expected result provided that we can prove that the logical structure obtained is a proof net.

4.3 Correction verification and chaining activation

In the second case mentioned above, we are not sure that the structure is a proof net. But rather than make an *a posteriori* verification (including a complete reconstruction in case of negative result), we prefer to make it during the construction. To do it, we construct, at the same time and step by step, $\mathcal{G}$ and a set $\mathcal{R}$ of proof nets that are fragments of $\mathcal{G}$.
At the beginning $\mathcal{R}$ is empty. But each new association produces an elementary net added to $\mathcal{R}$. Moreover the elements of $\mathcal{R}$ are nets opening on known nodes that are either inputs/outputs or connectors *times* or *par*. When two elements $\overline{\Pi_1}$ and $\overline{\Pi_2}$ in $\mathcal{R}$ open on the same *times* connector, we activate this one. Then $\overline{\Pi_1}$ et $\overline{\Pi_2}$ are replaced in $\mathcal{R}$ by a unique more complex net. Likewise, when an element $\overline{\Pi_0}$ of $\mathcal{R}$ has two edges opening on the same connector *par*, this one is activated and $\overline{\Pi_0}$ is replaced in $\mathcal{R}$ by the new net resulting of this activation. Each new association can thus start a chaining activation of binary connectors. During the construction, the elements of $\mathcal{R}$ are more and more complex. At the end, if the construction succeeds, $\mathcal{R}$ consists of a unique net: the expected one.

4.4 Incorrectness and backtracking

The construction of $\mathcal{R}$ is not necessarily linear. It is possible to create incorrect intermediate nets. For example, it can happen that an element of $\mathcal{R}$ element has two inputs/outputs opening

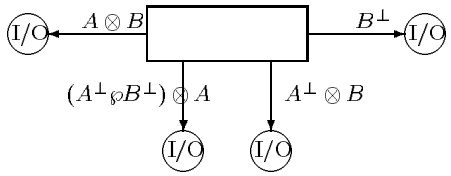
on the same connector *times*. In this case, we make a backtracking on the last association. This possibility of backtracking imposes us to conserve an important quantity of knowledge : for each element of $\mathcal{R}$ an history of its construction and an history of dual free edges of effective associations.

4.5 Duality property

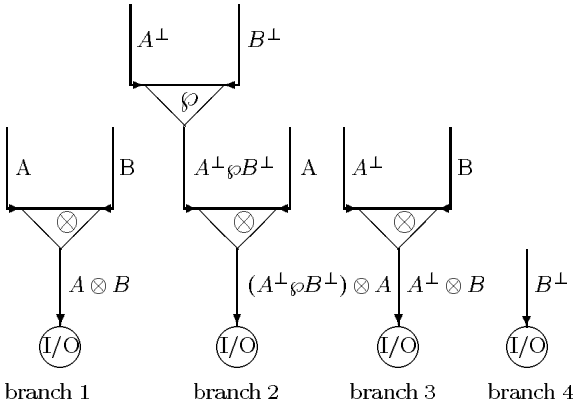
Before considering the net construction, it would be interesting to have simple criteria easy to verify that allow to filter certain classes of non provable sequents with a view to eliminating it directly. There exists one deduced property from the restricted framework of multiplicative linear logic named *duality property*. In a proof in multiplicative linear logic, we have a conservation of the atoms and the duality property is a consequence of this fact.
Duality Property: *If a sequent is provable in multiplicative linear logic then the multi-set of these atoms can be split into pairs of dual atoms.*

5 An example

Before to present the algorithm and its proofs, we want ot illustrate it through an example.
We aim to prove the sequent: $\vdash A \otimes B, (A^{\perp} \wp B^{\perp}) \otimes A, A^{\perp} \otimes B, B^{\perp}$.
It corresponds to the search of a proof net that would be the internal structure of the following box:

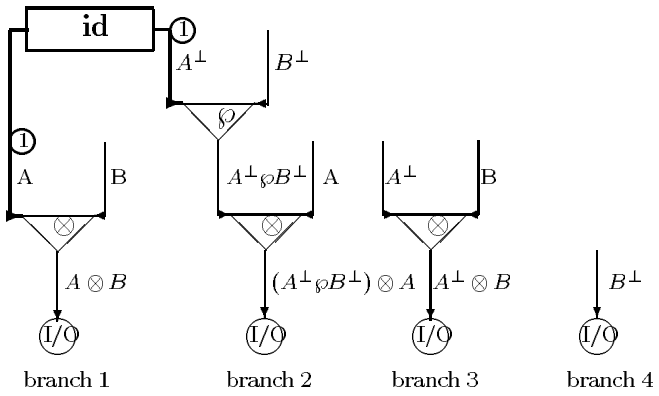


The sequent verifies the *duality property* and then we can begin the proof net construction. Let us consider the terminal branches

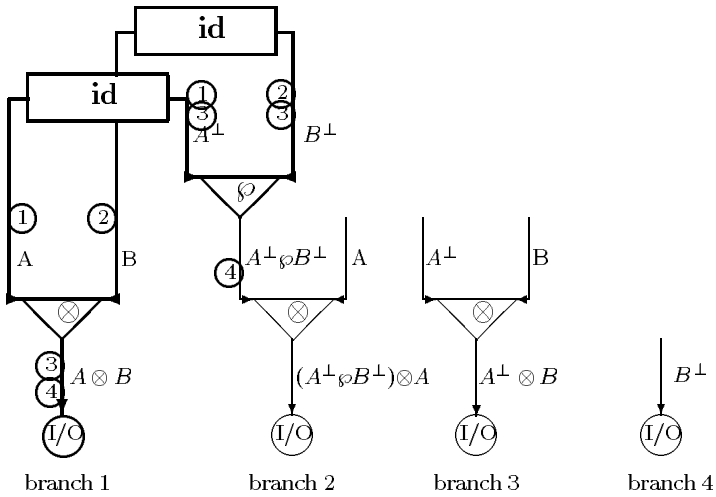


Let us choose the branch 1 as starting point of the construction of $\mathcal{G}$. It has two free edges (A and B) and we decide to associate the edge A. There are two possibilities: edge $A^{\perp}$ of branch 2 or of branch 3. Let us consider the edge $A^{\perp}$ of branch 2 for the association with A. A first consequence of this association is to extend $\mathcal{G}$ to branch 2. A second one is the creation of the

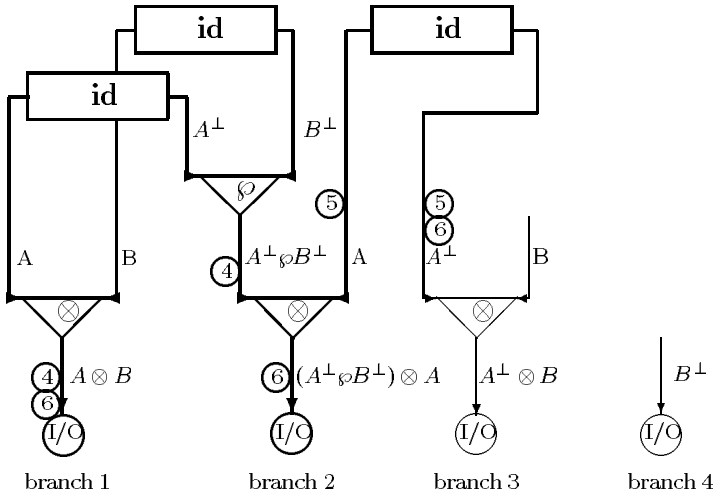
elementary net ①(to indicate it in the figure we numerate their terminal edges). This new proof net is the first element of the set $\mathcal{R}$.
Hence, the figure below (intermediate proof nets created during the construction are in bold)



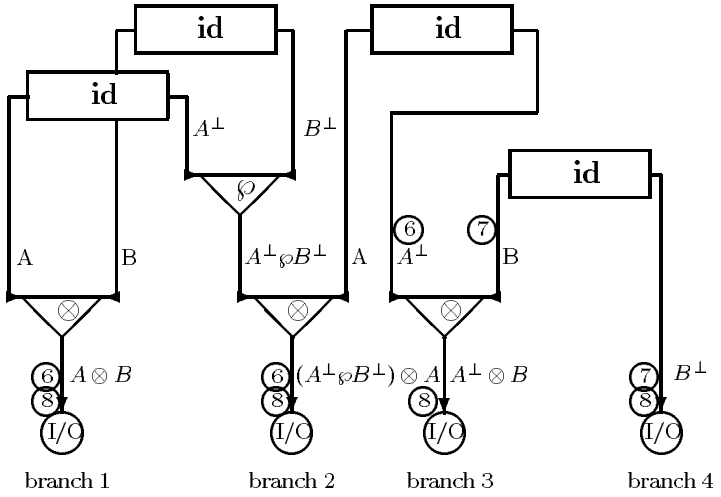
$\mathcal{G}$ now constituted of branches 1 and 2 together with the net ①, has three free edges. Let us choose the edge B of branch 1 and associate it, for example, with the edge $B^\perp$ of branch 2. We note ②this new elementary net that we add to $\mathcal{R}$. Then $\mathcal{R} = \{①, ②\}$.
We are trying a *chaining connection* of elements of $\mathcal{R}$ from the edge B (just considered) and then from $B^\perp$. The function *connections-propagation* realizes this task in the algorithm proposed in the next section. Seeing that B opens on a *times* connector, that is the arrival of the net ①of $\mathcal{R}$, this connector can be activated. It means we gather the nets ①and ②to constitute a new one ③. We verify that the connection is correct because the nets ①et ②are independent. Then we can replace in the set $\mathcal{R}$ the nets ①and ②by ③. Then $\mathcal{R} = \{③\}$.
The successor node of current node *times* is an input-output then the chaining connection from the edge B from branch 1 stops. We attempt the same process from the previous free edge $B^\perp$ of branch 2. The connector *par* of branch 2 can now be activated. Let us connect two terminal edges of net ③ we obtain the net named ④. We verify that this creation is correct, considering that the connected edges belong to the same net. Then we can replace in $\mathcal{R}$ the net ③by the net ④. The successor of the current node *par* is a *times* with a free initial edge then the chaining connection stops. Hence the following figure:



Now $\mathcal{G}$ has only one free edge: the edge A of branch 2 that can only be associated with the edge $A^\perp$ of branch 3. This association leads to the creation of the elementary net ⑤that is added to the set $\mathcal{R}$. Then $\mathcal{R} = \{④, ⑤\}$. The connector *times* of branch 2 can then be actived because the two initial edges are connected and it allows the union of nets ④and ⑤to give a new net ⑥that replaces it in the set $\mathcal{R}$. Then $\mathcal{R} = \{ ⑥\}$. Moreover, $\mathcal{G}$ is now extended to branch 3. Hence the figure:



We have now to associate the edge B of the branch 3 with the edge $B^\perp$ of the branch 4. We then create an elementary net ⑦ that will be added to $\mathcal{R}$. Then $\mathcal{R} = \{ ⑥, ⑦\}$.
The connector *times* of the branch 3 can be actived and allows the union of the nets ⑥and ⑦ resulting in a new net ⑧that takes their place in $\mathcal{R}$. Then $\mathcal{R} = \{ ⑧\}$. The graph $\mathcal{G}$ is now extended to the branch 4 and there is no more free edge. Consequently, the construction is terminated and is successful because there is no more terminal branch not connected to $\mathcal{G}$. The final net has the following form:



6 Proof net construction algorithm

6.1 Algorithm presentation

Definition 6.1 *The following definitions are function or predicate definitions used in the proof net construction algorithm:*
1) The set duality-test(A) *is the set of free edges $A^{\perp}$ that have already been tested for the association with* A.
2) The set assoc-edges *is the set of the edges that have been associated.*
3) The set $\mathcal{R}$ is the set of the (already constructed) intermediate nets. 4) The graph $\mathcal{G}$ is the net under construction.
5) The set term-branch($\vdash \Gamma$) *is the set of the terminal branches of $\vdash \Gamma$.*
6) duality($\vdash \Gamma$) *is true if the sequent $\vdash \Gamma$ verifies the duality property.*
7) exist-free-edge(S) *is true if there exists a free edge in S.*
8) property-free-edge(E) *is true if E is a free edge leading, if possible, to a binary connector that is the opening of a net from $\mathcal{R}$ and situated as high as possible in $\mathcal{G}$.*
9) good-link(A,$A^{\perp}$) *is true if the union of $\mathcal{G}$ (including A) and the terminal branch of B including $A^{\perp}$ has another free edges when B is not reduced to this union.*
10) new-graph($\mathcal{G}$,A,$A^{\perp}$)*is the graph obtained from $\mathcal{G}$ by associating the open edge A of $\mathcal{G}$ with the open edge $A^{\perp}$ of $\mathcal{G}$ or B.*
11) elem-proof-net(A,$A^{\perp}$) *is the new elementary proof net constructed through the association of A and $A^{\perp}$.*

The algorithm uses the intermediate function *connections-propagation*, the specification of which is given below:

function connections-propagation

> **inputs**: a net $\mathcal{G}$ under construction,
> a set $\mathcal{R}$ of intermediate already constructed proof nets,
> a node *current-node* of $\mathcal{G}$.
>
> **output**: an incorrectness message or a set $\mathcal{R}'$ of intermediate proof nets.
>
> **specification**:
> Activation by chaining the binary connectors of $\mathcal{G}$ from the node *current-node* following the order determined by the edges of $\mathcal{G}$ and going as far as possible.
> If, during this chaining connection, we detect an incorrectness, the function returns a message giving it else $\mathcal{R}'$ is the result of the successive activations of "times" and "par".

A complete presentation of this function is given in appendix A.

function net-construction

input: a sequent $\vdash \Gamma$ of multiplicative linear logic.

output: a proof net $\overline{\Pi}$ or a failure message

specification: If $\vdash \Gamma$ is provable in linear logic then $\overline{\Pi}$ is a proof net of $\vdash \Gamma$ else the function. returns a failure message

begin

if not *duality-property*($\vdash \Gamma$) **then return** "failure" **endif**
B := *term-branch*($\vdash \Gamma$);
for each free edge A of terminal branches in B
do *duality-test(A)* := $\emptyset$ **endfor**;
assoc-edges := $\emptyset$;
$\mathcal{R} := \emptyset$;
$\mathcal{G}$:= one of the terminal branches of B; $B := B$ - $\{\mathcal{G}\}$;
while *exist-free-edge*($\mathcal{G}$)
do choose a free edge A of $\mathcal{G}$ with *property-free-edge*(A);
nosuccessassoc := true;
while there exists in the graphs in B a free edge $A^{\perp} \notin$ *duality-test(A)* such that *good-link*(A,$A^{\perp}$) **and** nosuccessassoc.
do choose a free edge $A^{\perp}$ of $B \notin$ *duality-test*(A) such that *good-link*(A,$A^{\perp}$) **and** *property-free-edge*($A^{\perp}$);
duality-test(A) := $\{A^{\perp}\} \cup$ *duality-test*(A);
$\mathcal{G}$:= *new-graph*($\mathcal{G}$,A,$A^{\perp}$); update(B);
$\mathcal{R} := \mathcal{R} \cup \{$*elem-proof-net*($A$,$A^{\perp}$)$\}$;
$\mathcal{R}$:= **connections-propagation**($\mathcal{G}$, $\mathcal{R}$, A);
if $\mathcal{R} \neq$ incorrectness
then $\mathcal{R}$:=**connections-propagation**($\mathcal{G}$,$\mathcal{R}$,$A^{\perp}$)
endif
nosuccessassoc := ($\mathcal{R}$=incorrectness);
if nosuccessassoc
then give back to $\mathcal{G}, \mathcal{B}, \mathcal{R}$ the initial values before the association attempt of A with $A^{\perp}$
endif
endwhile
if not(*exist-free-edge*($\mathcal{G}$)) **and** card($\mathcal{R}$) $\neq 1$
then nosuccessassoc := true
endif.
if nosuccessassoc
then *duality-test*(A) := $\emptyset$;
if *assoc-edges* = $\emptyset$
then return "failure"
else extract the head A' of *assoc-edges* ;
Give back to $\mathcal{G}, \mathcal{B}, \mathcal{R}$ the initial values before the association attempt of A' with $A^{\perp}$
endif
else *assoc-edges* := cons(A ,*assoc-edges*)
endif
endwhile
return the unique net $\overline{\Pi}$ element of $\mathcal{R}$;

end

6.2 Proofs of the algorithm

In this section we want to present the different proofs of the algorithm that are the *termination*, *correctness* and *completeness* proofs.

6.2.1 Termination proof

This proof consists in proving the termination by associating with each loop in the algorithm a function with parameters characterizing the execution state of the algorithm. The value of the function will strictly decrease during the execution of the loop.
A complete presentation of this proof is given in appendix B.

6.2.2 Correctness proof

Proposition 6.1 *the function* net-construction*, if it succeeds, returns a net* $\overline{\Pi}$*, being correct and constituting a proof of the sequent* $\vdash \Gamma$ *given as input.*

Proof 6.1 *1) Let us prove that* $\overline{\Pi}$ *is correct.*
It is sufficient to prove that the set $\mathcal{R}$ *of intermediate created nets is, at any time of the execution of the algorithm, a set of correct nets.*
Let $\mathcal{R}_k$ *be the value of* $\mathcal{R}$ *after the execution of the* k^{th} *step of the algorithm. Because of the termination of the algorithm, the sequence of values of* $\mathcal{R}$ *is finite and can be written* $\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_n$ *where* $\mathcal{R}_0$ *is* $\emptyset$ *and* $\mathcal{R}_n = \{\overline{\Pi}\}$ *(considering that we are in the case where construction succeeds).*
Property: After the execution of the k^{th} step, the resulting set $\mathcal{R}_k$ is a set of correct sets.
proof*: by induction on k.*

- *if* $k = 0$*, we have* $\mathcal{R}_0 = \emptyset$ *and the property is true.*
- *Let us assume the property to be true for all* $k < n$*, and show that it is true for k+1.*
  If there is no modification of R_k *during the execution of* k^{th} *step, it is trivial else the modification can be of four types:*
  - *when the association of two dual free edges A et* $A^{\perp}$ *succeeds, an elementary net is added to* $\mathcal{R}_k$*.*
  - *when the association of two dual free edges A et* $A^{\perp}$*, there is backtracking on a previous value* $\mathcal{R}_h$ $(h < k)$ *of* $\mathcal{R}$*.*
  - *when the activation is on a* times *connector, there is a replacement in* $\mathcal{R}_k$ *of two nets by their union with this connector.*
  - *when the activation is on a* par *connector, there is a replacement in* $\mathcal{R}_k$ *of a net by another obtained by connecting two inputs-outputs of the former one with* par*.*

  It is clear that, as a result of one of these four modifications, $\mathcal{R}_{k+1}$ *is a set of correct nets. Then the property is true for k+1.* □

Then, the property is true for all $k \leq n$ *particularly for n. Consequently* $\overline{\Pi}$ *is a correct net.*

2) Let us prove that $\overline{\Pi}$ *is a proof of* $\vdash \Gamma$*, i.e., the set of formulae constituting the inputs-outputs of* $\overline{\Pi}$*, is the set of the formulae of* $\vdash \Gamma$*. At first, we prove that each input-output of* $\overline{\Pi}$ *is a formula of* $\vdash \Gamma$*. As* $\mathcal{G}$ *is connected and has its inputs-outputs coinciding with the formulae of* $\vdash \Gamma$ *we have the result.*

6.2.3 Completeness proof

Before beginning the completeness proof, we need to introduce the notion of construction-trace of a net and two propositions that will be necessary to obtain the completeness proof.

Definition 6.2 *A trace-construction of a proof net* $\overline{\Pi}$ *is a finite sequence* $\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_n$ *of net sets that verifies:*
- $R_n = \{\overline{\Pi}\}$
- for all k such that $0 \leq k < n$, $\mathcal{R}_{k+1}$ *is obtained from* $\mathcal{R}_k$ *by the connection of two elements of* $\mathcal{R}_k$ *with a* times *or the connection of two terminal edges of an* $\mathcal{R}_k$ *element with* par.

Proposition 6.2 *If* $(\mathcal{R}_k)_{0 \leq k < n}$ *is a trace-construction of a proof net* $\overline{\Pi}$ *and if there exists h such that* $0 \leq h < n$ *and* $\mathcal{R}_{h+1}$ *obtained from* $\mathcal{R}_h$ *by activation of a* par *on two edges that were terminal edges of a net of* $\mathcal{R}_0$, *then there exists a trace-construction* $(\mathcal{R}'_k)_{0 \leq k < n}$ *of* $\overline{\Pi}$ *such that* $\mathcal{R}'_0 = \mathcal{R}_0$ *and* $\mathcal{R}'_1$ *are obtained from* $\mathcal{R}'_0$ *by activation of the considered operator* "par" .

Proof 6.2 *By induction on h.*

Proposition 6.3 *If* $(\mathcal{R}_k)_{0 \leq k < n}$ *is a trace-construction of a proof net* $\overline{\Pi}$ *and if there exists h such that* $0 \leq h < n$ *and* $\mathcal{R}_{h+1}$ *obtained from* $\mathcal{R}_h$ *by activation of a* times *on two edges that were terminal edges of two distinct nets of* $\mathcal{R}_0$, *then there exists a trace-construction* $(\mathcal{R}'_k)_{0 \leq k < n}$ *of* $\overline{\Pi}$ *such that* $\mathcal{R}'_0 = \mathcal{R}_0$ *and* $\mathcal{R}'_1$ *are obtained from* $\mathcal{R}'_0$ *by activation of the considered operator* "times".

Proof 6.3 *By induction on h.*

Definition 6.3 *A set of generator nets of a proof net* $\overline{\Pi}$ *is a part of the first term of a trace-construction of* $\overline{\Pi}$.

Proposition 6.4 *(completeness) If* $\vdash \Gamma$ *is a provable sequent in the multiplicative linear logic then the function net-construction applied to* $\vdash \Gamma$ *returns a proof net of* $\vdash \Gamma$.

Proof 6.4 *Let us reason by refutation. We assume that the function* net-construction *returns a failure message and want to show that it leads to a contradiction.*
Let Π *a particular proof net of* $\vdash \Gamma$ *and n the number of the identity axioms it contains (n is strictly positive and depends only on* $\vdash \Gamma$*).*
Let us prove the following property P(k) for $0 \leq k < n$*:*
Property P(k)*:* At a certain time during the execution of the algorithm *net-construction*, one succeeds in associating k free edges, and $\mathcal{R}$ is then a set of generators net of $\overline{\Pi}$.
proof*:*
1) P(0) is true.
At the beginning of the execution, we do not try to associate a free edge with another one and $\mathcal{R}$ *is empty.*
2) Let us assume P(k) is true for $0 \leq k < n$ *and prove that P(k+1) is true.*
Let us consider the time when one has succeeded in associating k free edges and then $\mathcal{R}$ *is a set of generator nets of* Π *(it is possible because P(k) is true).*
$k < n$ *and free edges remain in* $\mathcal{G}$ *because* $\mathcal{G}$ *is never closed while all terminal branches are not linked. Let A be the free edge that is going to be selected by the algorithm and* $A^{\perp}$ *the one*

to which A is linked by an identity axiom in Π. $\mathcal{R}$ *is a set of generator nets of* Π *and* $A^\perp$ *is necessarily a free edge at this time.*
By hypothesis, the algorithm fails to construct a proof net of $\vdash \Gamma$ *and then all the eventual associations of A with a dual edge are going to fail; it implies that, at one time, we attempt to associate A with* $A^\perp$.
Let us consider the time when A and $A^\perp$ *are linked by an identity axiom and when we call the function connections-propagation for A.*
We have to prove that the propagation succeeds and that the resulting $\mathcal{R}$ *is a set of generator nets of* Π.
Let p the number of iterations of loop 3) that will be executed, we can prove by induction for $0 \leq h \leq p$ *the following property Q(h):*
Property Q(h)*:* After h executions of loop 3) there is no incorrectness and the set $\mathcal{R}$ remains formed with generators of Π.
proof*:*
1) Q(0) is true.
At call of connections-propagation, there is no incorrectness founded and from the induction hypothesis relative to P, $\mathcal{R}$ *is composed of generators of* Π.
2) Let us assume that Q(h) true for $0 \leq h < p$ *and let us prove that Q(h+1) is true.*
Considering the beginning of the (h+1)th execution of the loop 3), two cases can be considered:
a) current-node trains on a par *connector.*
If $\Pi_1 = \Pi_2$ *then* Π_1 *is replaced in* $\mathcal{R}$ *by* Π_1' *obtained by activation of this connector par. The set* $\mathcal{R}$ *becomes* $\mathcal{R}'$. $\mathcal{R}$ *being a set of generators of* Π, *by the induction hypothesis relative to Q, one can then, from the proposition 6.1, consider the activation of par as a first step of the construction of* Π *from* $\mathcal{R}$. *Consequently,* $\mathcal{R}'$ *is a set of generators of* Π.
If $\Pi_1 \neq \Pi_2$, *there is no incorrectness founded because by induction hypothesis relative to* $\mathcal{G}$, Π_1 *and* Π_2 *are generator nets of* Π.
b) node-current trains on a connector "times"
Π_1 *and* Π_2 *cannot be identical because by induction hypothesis related to* $\mathcal{G}$ *they are generator nets of* Π *and no incorrectness can be found. Moreover,* Π_1 *and* Π_2 *are replaced in* $\mathcal{R}$ *by* Π_{12} *obtained by activation of the times pointed. The set* $\mathcal{R}$ *becomes then* $\mathcal{R}'$. $\mathcal{R}$ *being a set of generators of* Π, *by the induction hypothesis relative to Q, one can then, from the proposition 6.2, consider the activation of times as a first step of the construction of* Π *from* $\mathcal{R}$. *Consequently,* $\mathcal{R}'$ *is a set of generators of* Π.
Then Q(h+1) is true and Q(h) is true for all h such that $0 \leq h \leq p$.
□
Particularly Q(p) is true and then the propagation of the connections from A has succeeded and the resulted $\mathcal{R}$ *is a set of generator nets of* Π.
We can do the same proof for the connections propagation from $A^\perp$.
The association of A and $A^\perp$ *has succeeded and the set* $\mathcal{R}$ *is a set of generators of* Π.
Then P(k+1) is true.
We proved that P(k) is true for all k such that $0 \leq k \leq n$.
□ *P(n) is true and consequently the algorithm succeeds for the construction of* Π.
That is in contradiction with the initial hypothesis.

To conclude, if $\vdash \Gamma$ *is provable then the algorithm succeeds in constructing a proof net for this sequent.*

7 Conclusion

We have shown we can construct a proof net automatically for a given sequent in multiplicative linear logic. This direct way to answer if a sequent is provable is a way to express the decidability of multiplicative linear logic. From this study we have a better view of the concept of proof net in linear logic and we can consider some applications with it as logical framework.
The connections between proof nets and proof-search algorithms for matrix-methods (connections [5] or matings [3]) would be studied and this algorithm could be helpful for such a work. Linear logic seems an adequat framework to consider plans generation from a logical point of view [15]. If we consider the conjonctive planification using MLL with proper axioms, a plan corresponds to a proof net. The direct extension of the algorithm for the proper axioms treatment is possible but presents some difficulties because of the cuts on these axioms. Considering logic programming, we can extend Prolog in the framework of MLL, in the same spirit as [2], but the expressivity gain is not obvious and moreover $\otimes$ sets some performance problems. But some works, like [13], emphasize the necessity to extend the work, for logic programming application, to additive and multiplicative linear logic (AMLL) and even to complete linear logic (LL). Then if we do not want to restrict our point to classical proof nets in MLL but to consider proofs in AMLL or LL, we have two possibilities. The first one is to extend the proof net notion to AMLL [4] and to abord its automated construction by a similar approach. Let us note that a direct extension of the presented algorithm appears problematic because it will not consist only in connecting open edges and considering the additive connectors terminal branches are not given a priori with the conclusion $\vdash \Gamma$. A second one is to consider the problem of proof construction directly in AMLL with a specific, and completely different, decision procedure [8] and to study its relationship with extension of proof nets notion. Even so, the mechanization of proof net construction in MLL, presented here, is a first attempt for the use of proof net and linear logic, having in mind the computational interpretation and its applications.

References

[1] S. Abramsky. Computational interpretations of linear logic. technical report, Department of Computing, Imperial College, London SW7 2Bz, England, 1991.

[2] J.M. Andreoli and R. Pareschi. Logic programming with sequent systems: A linear logic approach. In *Int. Workshop on Extensions of Logic Programming, LNCS 475*, pages 1–30, Tübingen, Germany, December 1989.

[3] P. Andrews. Theorem proving via general mating. *Journal of ACM*, 28(2):193–214, 1981.

[4] G. Bellin. Proof nets for multiplicative and additive linear logic. Technical Report ECS-LFCS 91-161, Department of Computer Science, Edinburgh University, May 1991.

[5] W. Bibel. On matrices with connections. *Journal of ACM*, 28(4):633–645, 1981.

[6] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[7] D. Galmiche. Constructive system for automatic program synthesis. *Theoretical Computer Science*, 71(2):227–239, 1990.

[8] D. Galmiche and G. Perrier. Automated deduction in additive and multiplicative linear logic. To appear in *Logic at Tver '92, Symposium on Logical Foundations of Computer Science*, Tver, Russia, July 1992.

[9] G. Gentzen. Collected papers. *Edited by M. E. Szabo, Amsterdam*, 1969.

[10] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[11] J.-Y. Girard. Towards a geometry of interaction. In J.-W. Gray and A. Scedrov, editors, *AMS Conference on categories in computer science and logic*, pages 69–108, Boulder-Colorado, June 1987.

[12] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[13] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *6th IEEE symposium Logic in Computer Science*, pages 32–42, Amsterdam, July 1991.

[14] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *31st annual IEEE Symp. on Foundations of Computer Science*, St-Louis, Missouri, October 1990.

[15] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science, LNCS 472*, pages 63–75, Bangalore, India, December 1990.

[16] U. Solitro. A typed calculus based on a fragment of linear logic. *Theoretical Computer Science*, 68:333–342, 1989.

[17] A.-S. Troelstra and D. Van Dalen. *Constructivism in Mathematics, an Introduction*. North-Holland, Amsterdam, 1988.

A The function connections-propagation

The algorithm uses the auxiliary function *connections-propagation* whose specification is given below:

function connections-propagation

inputs: a net $\mathcal{G}$ under construction,
a set $\mathcal{R}$ of intermediate already constructed nets,
a node *current-node* of $\mathcal{G}$.

output: an incorrectness message or a set $\mathcal{R}'$ of intermediate proof nets.

specification: Activation by chaining the binary connectors of $\mathcal{G}$ from the node *current-node* following the order determined by the edges of $\mathcal{G}$ and going as far as possible.
If, during this chaining connection, we detect an incorrectness, the function returns a message giving it, else $\mathcal{R}'$ is the result of the successive activations of "times" and "par".

Definition A.1 *i)* connector-link(N) *is true if N is a connector linking two edges of two nets* $\overline{\Pi_1}$ *and* $\overline{\Pi_2}$ *of* $\mathcal{R}$ *(not necessarily distinct).*
ii) activation(c,n) *gives as result the net obtained by activation of the connector c of node n, conserving the history of the net construction.*
iii) succ(n) *is the arriving node corresponding to the edge starting from n.*

function connections-propagation ($\mathcal{G}$ **: net;**$\mathcal{R}$ **: set of nets; current-node : node).**
begin
propagation := true
while connector-link(*current-node*) **and** propagation
do **if** *current-node* is a "par"
then if $\overline{\Pi_1} = \overline{\Pi_2}$
then $\mathcal{R} := \mathcal{R}$ - $\{\overline{\Pi_1}\} \cup$ activation("par", *current-node*);
current-node := succ(*current-node*)
else propagation := false;
if $\overline{\Pi_1}$ and $\overline{\Pi_2}$ do not have a terminal edge linked to a connector "times"
then return "incorrectness"
endif
endif
else if $\overline{\Pi_1} = \overline{\Pi_2}$
then return "incorrectness"
else $\mathcal{R}$:=$\mathcal{R}$-$\{\overline{\Pi_1}, \overline{\Pi_2}\} \cup$ activation("times",*current-node*);
current-node := succ(*current-node*)
endif
endif
endwhile
return($\mathcal{R}$)
end

B Termination proof

The algorithm for the **net-construction** function has four *while* loops:

- the loop 1 (iteration) attempting to associate a free edge of $\mathcal{G}$ with another one;
- the loop 2 attempting to associate a free edge $A^{\perp}$ with a given free edge A;
- the loop 3 for the activation of the binary connectors following a given edge A;
- the loop 4 for the activation of the binary connectors following a given edge $A^{\perp}$.

We will prove the termination of the algorithm by associating with each loop a function on integer with parameters characterizing the execution state of the algorithm. The value of the function will strictly decrease during the execution of the loop.

- Let us consider the loops 3 and 4.
  We associate the function that returns, at each iteration, with the number of binary connectors following the value of the *current-node*. This function decreases of 1 at each iteration, that proves the termination of these loops.

- Let us consider the loop 2.
  We associate it with the function that returns the number of free edges $A^{\perp}$ that are not member of *duality-test(A)*. This function decreases (of 1) at each iteration, which proves the termination of the loop 2.

- Let us finally consider the loop 1.
  The function to associate is less evident to determine because of the backtracking on edges associations (soon realized) Let $A_1, A_2, ..., A_p$ be the finite sequence of free edges that the algorithm attempt to associate in the order where they are met for the first time.
  We associate, for each step of execution, with each edge A_i an integer α_i, that represents the number of free edges $A_i^{\perp}$ not member, at this time, of *duality-test(A_i)*. The set of α_i is upper bounded by a number independently of the step where we are in the algorithm. Let us note n such an upper bound.
  Let us consider the function φ to which corresponds, at each execution step, the value: $\sum_{i=1}^{p} \alpha_i n^{n-i}$.
  We have to study it along the execution. Let us consider the beginning of an iteration where the edge to associate is A_k $(1 \leq k \leq n)$. At the end of the iteration, two situations are possible:

  - either the association successes and the value of the function φ decreases of n^{n-k} that is strictly positive;
  - or the association fails and we have to do a backtracking. After a finite number of iterations that will follow, either the algorithm terminate with a definitive failure, or we succeed in obtaining a new association of an edge A_h such that $h < k$. From the beginning of the association attempt of A_k to the end of the successful association of A_h. α_h has decreased of one and α_i, $h < i \leq k$, has again its maximal value and then is affected by a variation less than or equal at $n-1$.
    The function φ has supported a variation equal (at maximum) to :
    $-n^{n-h} + (n-1)\sum_{i=h+1}^{k} n^{n-i} = -n^{n-h} + (n-1)\frac{n^{n-h}-n^{n-k}}{n-1} = -n^{n-k} < 0.$

  The function φ does not strictly decrease at each iteration but its properties are sufficient to assert that the loop terminates.