A Linear-time Algorithm for Verifying MLL Proof Nets via Lamarche's Essential Nets

(Preliminary Version)

A. S. Murawski* C.-H. L. Ong†

Abstract

We consider the following decision problems:

PROOFNET : Given a multiplicative linear logic (MLL) proof structure, is it a proof net?

ESSNET : Given an essential net [9] of an intuitionistic MLL formula, is it correct?

In this paper we give a linear-time algorithm for ESSNET. By showing that PROOFNET is linearly reducible to ESSNET, we obtain a linear-time algorithm for PROOFNET. Hence the question of whether a given linearly balanced MLL formula is provable can be settled by a linear-time algorithm.

1 Introduction

By MLL we mean the classical multiplicative $(\otimes, \parr, -^{\perp})$-fragment of linear logic [7]. This paper is concerned with a fast algorithm that decides if a given MLL proof structure is a proof net, equivalently, if a given *linearly balanced* (i.e. each atom that appears in the formula has exactly one positive and one negative occurrence) MLL formula is provable.

MLL and Proof Nets

First a quick review of MLL. We let $a, b, c, \cdots$ range over propositional atoms. A *literal* is either a propositional atom a or its linear negation $a^{\perp}$. MLL formulas, ranged over by $A, B, C, \cdots$, are generated from the literals by the connectives $\otimes$ (tensor) and $\parr$ (par). The MLL Sequent Calculus is presented as follows: we let $\Gamma, \Delta, \cdots$ range over finite sequences of formulas

$$\textbf{(Id)} \quad \frac{}{\vdash a^{\perp}, a} \qquad\qquad \textbf{(Exch)} \quad \frac{\vdash \Gamma}{\vdash \sigma(\Gamma)}$$

$$\textbf{(Tensor)} \quad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \qquad\qquad \textbf{(Par)} \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$

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

*Oxford University Computing Laboratory (OUCL), Wolfson Building, Parks Road, Oxford OX1 3QD, UK. andrzej@comlab.ox.ac.uk

†Homepage: http://www.comlab.ox.ac.uk/oucl/people/luke.ong.html Email: Luke.Ong@comlab.ox.ac.uk

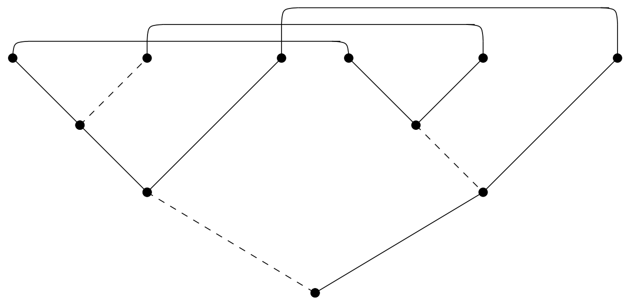


Figure 1: The DR-graph Gr($((a^{\perp} \parr b) \otimes b) \parr ((a \otimes b^{\perp}) \parr b^{\perp})$, ϕ, σ)

In the rule (**Exch**), $\sigma(\Gamma)$ means a permutation of the formulas in Γ. Note that (**Id**) applies only to propositional atoms. This simplifies our argument but does not affect provability. In this paper we are not concerned with (multiplicative) units.

A *proof structure* is a pair (Γ, ϕ) where Γ is a sequent and ϕ is a bijection between occurrences of atoms and occurrences of negated atoms in Γ such that ϕ maps an occurrence of an atom to an occurrence of the negation of the same atom. A *DR-switching* σ [5] for a proof structure (Γ, ϕ) is an assignment of L (left) or R (right) to each occurrence of $\parr$ in Γ. The associated *DR-graph* $\mathrm{Gr}(\Gamma, \phi, \sigma)$ is a graph that is obtained from the disjoint union of the respective abstract syntax trees of the formulas in Γ by performing the following edge operations:

- for each occurrence i of an atom, draw in an edge joining i and $\phi(i)$
- for each occurrence j of $\parr$, if $\sigma(j) = L$ (respectively R), delete the edge that joins j to its right (respectively left) child.

Example 1.1. Consider the (singleton) sequent $\Gamma = ((a^{\perp} \parr b) \otimes b) \parr ((a \otimes b^{\perp}) \parr b^{\perp})$. We give each occurrence of $b, b^{\perp}$ and $\parr$ in Γ a unique label as in

$$((a^{\perp} \parr_1 b_1) \otimes b_2) \parr_2 ((a \otimes b_3^{\perp}) \parr_3 b_4^{\perp}).$$

Set

$$\begin{aligned} \phi &= \{ (a, a^{\perp}), (b_1, b_3^{\perp}), (b_2, b_4^{\perp}) \} \\ \sigma &= \{ (\parr_1, L), (\parr_2, R), (\parr_3, R) \}. \end{aligned}$$

The DR-graph $\mathrm{Gr}(\Gamma, \phi, \sigma)$ is shown in Figure 1. Each dashed edge in the graph indicates an edge deleted from the abstract syntax tree of Γ.

Proof nets are a representation of the canonical proofs of linear logic introduced by Girard. As we are only concerned with provability and not proofs, it suffices to consider *normal* proof nets (see [7] for further details). Intuitively, a proof net is a proof structure that is correct i.e. one that is given by a proof. Girard's original correctness criterion in [7] has been simplified by Danos and Regnier [5] and we use their version here: we define a *proof net* to be a proof structure (Γ, ϕ) such that for every DR-switching σ, the associated DR-graph $\mathrm{Gr}(\Gamma, \phi, \sigma)$ is connected and acyclic.

Girard, Danos and Regnier have shown that an MLL sequent $\vdash \Gamma$ is provable if and only if there is some ϕ such that (Γ, ϕ) is a proof net. Thus if there are n occurrences of $\parr$ in Γ, to decide whether $\vdash \Gamma$ is provable, (for any ϕ) there are 2^n graphs to check for connectedness and acyclicity.

The reader may wish to verify that (Γ, ϕ) in Example 1.1 is a proof net; but (Γ, ϕ'), where $\phi' = \{ (a, a^{\perp}), (b_1, b_4^{\perp}), (b_2, b_3^{\perp}) \}$, is not.

In the following we shall only consider proof structures (P, ϕ) that are determined by *linearly balanced* (i.e. each atom that appears in P has exactly one positive and one negative occurrence) MLL *formulas* P, so that ϕ is redundant.

IMLL and Essential Nets

By IMLL we mean the intuitionistic multiplicative $(\otimes, \multimap)$-fragment of linear logic. IMLL formulas are generated from atoms $a, b, c, \cdots$ by the connectives $\otimes$ and $\multimap$ (linear implication). As before we let Γ, Δ range over finite sequences of IMLL formulas. The valid IMLL sequents are defined by the following rules:

$$\textbf{(var)} \quad a \vdash a \qquad\qquad \textbf{(exch)} \quad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

$$(\otimes\textbf{-l}) \quad \frac{A, B, \Gamma \vdash C}{A \otimes B, \Gamma \vdash C} \qquad\qquad (\otimes\textbf{-r}) \quad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$$

$$(\multimap\textbf{-l}) \quad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{A \multimap B, \Gamma, \Delta \vdash C} \qquad\qquad (\multimap\textbf{-r}) \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$\textbf{(cut)} \quad \frac{\Gamma, A \vdash B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

Given an IMLL formula A, we can define its classical counterpart $\overline{A}$ (by mutual recursion with $\underline{A}$) as follows:

$$\begin{array}{rclcrcl} \overline{a} & = & a & \qquad & \underline{a} & = & a^{\perp} \\ \overline{A \multimap B} & = & \underline{A} \mathbin{\rotatebox[origin=c]{180}{$\&$}} \overline{B} & & \underline{A \multimap B} & = & \overline{A} \otimes \underline{B} \\ \overline{A \otimes B} & = & \overline{A} \otimes \overline{B} & & \underline{A \otimes B} & = & \underline{A} \mathbin{\rotatebox[origin=c]{180}{$\&$}} \underline{B}. \end{array}$$

By a simple induction over the rules of the IMLL Sequent Calculus, we have the following:

Proposition 1.2. *For any IMLL formulas $C_1, \cdots, C_n$ and A, we have $C_1, \cdots, C_n \vdash A$ is IMLL-provable if and only if $\vdash \underline{C_1}, \cdots, \underline{C_n}, \overline{A}$ is MLL-provable.* □

Any linearly balanced IMLL formula A can be transformed to a variant MLL formula $\ulcorner A \urcorner$ which is constructed from *polarised* atoms and connectives by the following rules:

$$\begin{array}{rclcrcl} \ulcorner a \urcorner & = & a^{+} & \qquad & \llcorner a \lrcorner & = & a^{-} \\ \ulcorner A \multimap B \urcorner & = & \llcorner A \lrcorner \mathbin{\rotatebox[origin=c]{180}{$\&$}}^{+} \ulcorner B \urcorner & & \llcorner A \multimap B \lrcorner & = & \ulcorner A \urcorner \otimes^{-} \llcorner B \lrcorner \\ \ulcorner A \otimes B \urcorner & = & \ulcorner A \urcorner \otimes^{+} \ulcorner B \urcorner & & \llcorner A \otimes B \lrcorner & = & \llcorner A \lrcorner \mathbin{\rotatebox[origin=c]{180}{$\&$}}^{-} \llcorner B \lrcorner. \end{array}$$

The *essential net* [9] of A, written $\mathcal{E}(A)$, is a directed graph obtained from the abstract syntax tree of $\ulcorner A \urcorner$ by augmenting it by oriented axiom links (from a positive atom a^{+} to its negative counterpart a^{-}), and by drawing in oriented edges incident to the connective-nodes (namely $\mathbin{\rotatebox[origin=c]{180}{$\&$}}^{+}, \mathbin{\rotatebox[origin=c]{180}{$\&$}}^{-}, \otimes^{+}$ and $\otimes^{-}$) according to the *directional rules* in Figure 2.

Note that there is no edge between a $\mathbin{\rotatebox[origin=c]{180}{$\&$}}^{+}$-node and its negative left "child", which we shall call its *sink*. Henceforth by an essential net, we mean the essential net of some linearly balanced IMLL

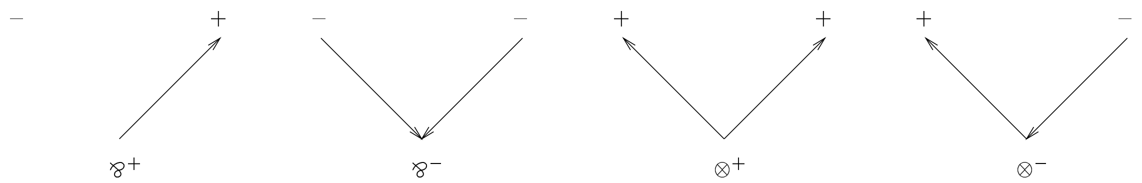


Figure 2: The directional rules

formula. For example the essential net of

$$f \otimes ((c \multimap (c \otimes f)) \multimap (e \otimes d)) \quad \multimap \quad (b \multimap (b \otimes e)) \otimes (a \multimap (a \otimes d))$$

(by convention $\otimes$ binds more strongly than $\multimap$) is presented in Figure 3.

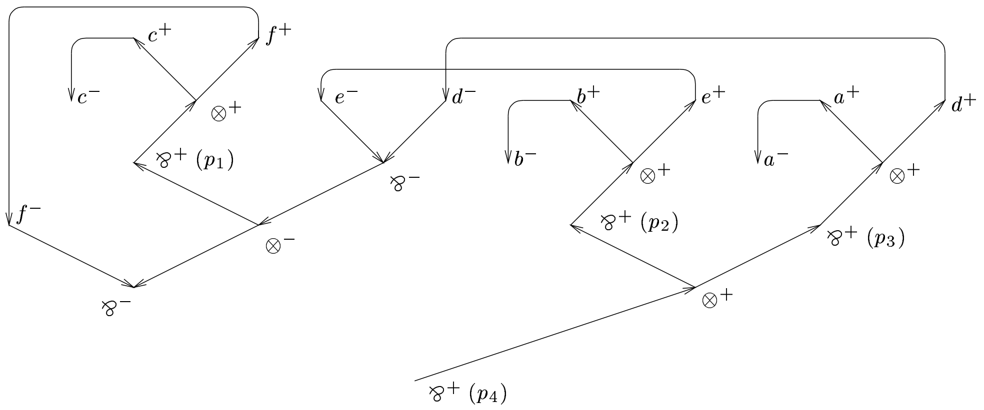


Figure 3: The essential net of $f \otimes ((c \multimap (c \otimes f)) \multimap (e \otimes d)) \ \multimap\ (b \multimap (b \otimes e)) \otimes (a \multimap (a \otimes d))$

An essential net is said to be *correct* [9] if it is acyclic, every node is reachable from the root, and every $\parr^+$-node p satisfies *Condition L*: every path from the root that reaches p's sink passes through p. An important result in [9] is a characterisation of IMLL provability:

Theorem 1.3 (Lamarche). *For any linearly balanced IMLL formula A, $\vdash A$ is provable if and only if $\mathcal{E}(A)$ is correct.*

Proof. We give a new proof of the Theorem in Appendix A. □

The reader may wish to check that the essential net in Figure 3 is correct.

Remark 1.4. In this paper, essential nets $G = \mathcal{E}(A)$ are by definition obtained from *linearly balanced* IMLL formulas A. For such nets (but not in general – consider $\mathcal{E}(a \multimap b)$), the reachability condition, i.e. every node is reachable from the root, is in fact a consequence of acyclicity. We claim that for every node of an acyclic essential net G, there is a backtracking path from it that reaches back to the root. Take any positive node; if it is not the root, then there is a unique edge that points to it, and so we can backtrack (uniquely) from it. Take any negative node. If it is a $\parr^-$-node, we can backtrack from it in two ways; if it is a $\otimes^-$-node we can backtrack from it in exactly one way; if it is some negative atom, the same is true because of the linearly balanced assumption. Since G is acyclic, every backtracking path must stop at some node, which is the root.

It is well-known that the problem of deciding the provability of a given MLL formula is NP-complete [10]. Here we shall consider provability decision problems restricted to linearly balanced formulas:

PROOFNET : Given an MLL proof structure, is it a proof net?

ESSNET : Given an IMLL essential net, is it correct?

In this paper we give a linear-time algorithm for ESSNET (Algorithm A in Figure 4). By showing that PROOFNET is linearly reducible to ESSNET, we obtain a linear-time algorithm for PROOFNET (Algorithm B in Figure 5). Hence the question of whether a given linearly balanced MLL formula is provable can be settled by a linear-time algorithm.

Guerrini [8] was the first to obtain a linear-time algorithm for PROOFNET. He uses a reformulation of Danos' contractibility criterion as a kind of unification problem. Though both Guerrini's algorithm and ours make use of a special linear-time case of the Disjoint Set Union Problem, it is not (yet) clear what the connexions between them are. We hope the reader will find our approach via Lamarche's essential nets simple and easy to implement. Work is underway to use the approach to design algorithms for deciding provability of larger fragments of linear logic.

2 An Algorithm for Verifying Essential Nets

First a quick introduction to the Disjoint Set Union Problem (see e.g. [4, Chap. 22]). The problem concerns three standard operations: we let v and v' range over a finite set of (input) elements

makeset(v) : construct a new singleton set $\{ v \}$ whose *name* (or representative) is v

find(v) : return the name of the set containing v

union(v, v') : construct[1] a new set that is the union of the respective sets containing v and v'; the new set is named by the name of the old set containing v.

Gabow and Tarjan [6] have shown that there is a linear-time algorithm for the special case where the structure of the unions (defined by a "union tree") is known in advance. They call the special case *Static Tree Set Union Problem* which can be stated as follows. We are given a rooted tree T of n nodes. Initially every node v of the tree is in a singleton set $\{ v \}$ named v, so that **find**$(v) = v$. We write the parent of v in T by $parent(v)$. We wish to perform on-line (i.e. each operation must be completed before the next one is known) an intermixed sequence of m **find** and **link** operations where **link**(v) is equivalent to **union**$(parent(v), v)$. The Gabow-Tarjan algorithm has time complexity $O(m + n)$. In the following, we shall make use of their algorithms for **find** and **link**.

Algorithm A

Fix an acyclic essential net G. Recall that given any finite directed graph, it is linearly decidable if it is cyclic. For convenience we assume that the root of G is a $\parr^+$-node. (In case the root of a graph is a $\otimes^+$-node, we split it into the two branches of the node, and if both components are still linearly balanced, we apply the Algorithm to each separately. See $\mathbf{A}'(G)$ in Figure 5 for a definition of the general case.) Let $p_1, \cdots, p_k$ be the $\parr^+$-nodes of G. We assume that p_k is the root. We define the relation $p_i \curvearrowright p_j$ to mean that there is a non-trivial path in G from p_i to p_j without passing through

[1] This operation assumes that v and v' are initially in different sets, and destroys the old sets containing v and v'.

Algorithm A

Input: an acyclic essential net G, with $\wp^+$-nodes $p_1, \cdots, p_k$ where p_k is the root of G

- Apply *Backtrack* to each $\wp^+$-node in any order such that the procedure is performed on a $\wp^+$-node only if it has been performed on all of its children. We can derive one such order from a depth-first search of G.

  Backtrack(p_i): *Visit*(s_i, i) where s_i is the sink of p_i.

  Procedure *Visit*(v, i) : **cases**

  1. $v = \text{root} \wedge i = k \;\rightarrow\;$ label v by i
  2. $v = \text{root} \wedge i \neq k \;\rightarrow\;$ **return no**
  3. v unlabelled $\wedge\; v = p_i \;\rightarrow\;$ label v by i
  4. v unlabelled $\wedge\; v = p_j \wedge j \neq i \;\rightarrow\;$ **return no**
  5. v unlabelled $\wedge\; v$ not a $\wp^+$-node $\;\rightarrow\;$ **do** *Visit*(u, i) **for** each predecessor u of v; label v by i
  6. v labelled by $i \;\rightarrow\;$ **stop**
  7. v labelled by some $j \neq i \wedge \textbf{find}(p_j) = p_i \;\rightarrow\;$ **stop**
  8. v labelled by some $j \neq i \wedge \textbf{find}(p_j) = p_l \wedge l \neq i \;\rightarrow\;$ *Visit*(u, i) where u is the predecessor of p_l; $\textbf{link}(p_l)$

- **return yes**

Figure 4: Algorithm A for EssNet

any other $\wp^+$-node. The relation defines an acyclic connected graph[2], though it is not in general a tree (see Example 2.1). Let T_G be *any* spanning tree of this graph. We can obtain one such directly from G in linear time by a depth-first search. Henceforth whenever we mention the ancestor, descendant, parent or child node of some $\wp^+$-node, it is with reference to a chosen T_G. We say that the node v_1 is a *predecessor* of v_2, or v_2 is a *successor* of v_1, if $v_1 \rightarrow v_2$ is an edge of the directed graph G.

We present the Algorithm in Figure 4. We write ; to mean sequential composition. Note that Algorithm A returns "no" as soon as a **return no** is executed in any run of *Visit*. We refer to Case 8 of the Algorithm as a *jump* (from the j-labelled v) to p_l.

Example 2.1. Take the provable IMLL formula

$$f \otimes ((c \multimap (c \otimes f)) \multimap (e \otimes d)) \quad \multimap \quad (b \multimap (b \otimes e)) \otimes (a \multimap (a \otimes d))$$

whose essential net we have given in Figure 3. There are four $\wp^+$-nodes, namely, p_1, p_2, p_3 and p_4, as indicated in parentheses in the Figure. The $\curvearrowright$-relation, which is a partial order, has the shape of a diamond, with p_4 at the top and p_1 at the bottom. Note that during *Backtrack*(p_2), no node labelled 1 was visited (for otherwise, Condition L would have to fail for p_2 as there would be a backtracking path from p_2's sink to the root via p_3), so that $\textbf{find}(p_1)$ is still p_1 at the end of *Backtrack*(p_2); similarly for *Backtrack*(p_3). However $\textbf{find}(p_i) = p_4$ for all i at the end of *Backtrack*(p_4).

[2]This is not necessarily true in case the root of G is a $\otimes^+$-node.

3 Correctness of Algorithm A

For any $\parr^+$-node p, Condition L is equivalent to: every backtracking path from p's sink reaches back to p. Note that at the start of each *Backtrack*(p_i), no node has label i. The effect of *Backtrack*(p_i) is to label certain nodes i; throughout Algorithm A no node is ever labelled more than once. The intuition is that the label i annotated to a node is a certificate that all backtracking paths from it reach back to p_i. Though the label, once given, is never changed, the name of each p_i (as given by $\mathbf{find}(p_i)$) can and does change from time to time during the course of Algorithm A, subject to the invariant that if v is labelled j and $\mathbf{find}(p_j) = p_l$ then every backtracking path from v reaches back to p_l (see (I1) below). The efficiency of the Algorithm is achieved by ensuring that the real work of checking each node is never duplicated.

We find it useful to identify several invariants for the correctness proof of Algorithm A. If v has a label j and $\mathbf{find}(p_j) = p_l$, we refer to l as the *name* of v.

Invariant for **Algorithm A**

(I1) If v is labelled j and $\mathbf{find}(p_j) = p_l$ then every backtracking path from v reaches back to p_l, passing only through nodes named l.

Invariants for *Backtrack*(p_i): During the execution of *Backtrack*(p_i)

(I2) whenever $\mathbf{find}(p_j)$ changes, the new name is p_i

(I3) if the label of v is j at the completion of *Visit*(v, i) then $\mathbf{find}(p_j) = p_i$.

Lemma 3.1. *Suppose Backtrack*(p_i) *has succeeded.*

(i) The first node for which Visit(v, i) *succeeds is always p_i. Therefore p_i is the first node to be labelled i.*

(ii) From the start of Algorithm A *until p_i is labelled,* $\mathbf{find}(p_i) = p_i$ *holds.*

(In fact (ii) above holds till the end of *Backtrack*(p_i) by (I2), but we cannot use (I2) yet – our induction hypotheses are instances of the invariant at all times before the start of *Backtrack*(p_i).)

Proof. Before the start of *Backtrack*(p_i), we have $\mathbf{find}(p_j) \neq p_i$ for each label j. This is because by (I3) at the moment of labelling, we have $\mathbf{find}(p_j) \neq p_i$; and by (I2) for every *Backtrack*(p_k) executed after that point but before the start of *Backtrack*(p_i), we also have $\mathbf{find}(p_j) \neq p_i$ whenever $\mathbf{find}(p_j)$ changed, because *Backtrack*(p_i) has not been invoked. This has two consequences. First $\mathbf{link}(p_i)$ has never been invoked and so $\mathbf{find}(p_i) = p_i$ at all times up to the start of *Backtrack*(p_i). Secondly Case 7 never happened, so the only cases that could lead to the first success of *Visit*(v, i) are 1 or 3 i.e. $v = p_i$. Also $\mathbf{find}(p_i) = p_i$ when p_i is labelled. □

We verify the invariants immediately after their parameters have changed. Thus we need to check the following cases:

(I1)	1, 3, 5, 8
(I2)	8
(I3)	1, 3, 5, 6, 7, 8

Case 1, 3 : No **find** value has changed since the start of *Backtrack*(p_i) (as in the proof of Lemma 3.1, Case 7 never happened) so (I1) holds for all nodes that are distinct from $v = p_i$. p_i is the only node with label i and, we have $\mathbf{find}(p_i) = p_i$ by Lemma 3.1, so (I3) holds. (I1) is trivially satisfied by $v = p_i$.

Case 5 : To verify (I3) for v we have to check that $\mathbf{find}(p_i) = p_i$, which follows from the Lemma 3.1 (it held at the start of *Backtrack*(p_i)) and (I2) (it has not changed). By (I3) and (I1) for the predecessors of v, we have (I1) for v, because each backtracking path from v must go through one of its predecessors.

Case 6 : By (I3) at the time v was labelled and (I2), we have $\mathbf{find}(p_i) = p_i$; hence (I3) holds.

Case 7 : (I3) is satisfied trivially, because the assumption is $\mathbf{find}(p_j) = p_i$.

Case 8 : By (I3) for *Visit*(u, i), if u has label h, $\mathbf{find}(p_h) = p_i$, and by (I1) at that moment, each backtracking path reaches back to p_i passing only through nodes with name i. Therefore $\mathbf{find}(\mathit{parent}(p_l)) = p_i$ (as *parent*(p_l) lies on one such path) and $\mathbf{link}(p_l)$ changes $\mathbf{find}(p_l)$ to p_i as required in (I2). Also $\mathbf{find}(p_j) = p_i$ thus satisfying (I3).

The operation $\mathbf{link}(p_l)$ changes the name of every node previously named l to i (the rest remain unchanged). By (I1) immediately before $\mathbf{link}(p_l)$, each backtracking path from such a node reaches back to p_l, passing only through nodes named l; and by (I1) applied at the moment *Visit*(u, i) is completed, any backtracking path from u reaches back to p_i passing only through nodes with name i. Thus each backtracking path from any node named l before $\mathbf{link}(p_l)$ is performed must reach back to p_i; and, after $\mathbf{link}(p_l)$ is performed, it can only pass through nodes named i. Thus (I1) holds for the nodes whose names have changed.

For each i, when *Backtrack*(p_i) succeeds, p_i's sink must be named i by (I3), and by (I1) it satisfies Condition L. Therefore if Algorithm A returns **yes**, Condition L holds.

Lemma 3.2. *Suppose Backtrack applied to all of p_i's descendants has succeeded. If Backtrack(p_i) fails, then p_i does not satisfy Condition L.*

Proof. The precondition for executing *Backtrack*(p_i) is that *Backtrack* has succeeded for every $\wp^+$-node which is a child of p_i. This implies that every $\wp^+$-node that can *only* be reached from the root via p_i has been labelled. Thus if an unlabelled $\wp^+$-node p_j such that $j \neq i$ is visited during *Backtrack*(p_i), there is a backtracking path from p_i's sink that reaches back via p_j to the root without passing through p_i, which is a violation of Condition L. For the other case of visiting the root when $i \neq k$ we need to check that p_i was not bypassed as a result of a jump. Suppose it was, it would mean that p_i had been labelled as a result of some earlier, successful *Backtrack*(p_j), which would be a contradiction because p_i was unlabelled when visited during *Backtrack*(p_j). □

Lemma 3.3. *Algorithm A always terminates.*

Proof. Since G is acyclic, non-termination can only arise because of jumps. So suppose during *Backtrack*(p_i), jumping from a j-labelled node v to $\mathbf{find}(p_j) = p_l$ where $l \neq i$ and backtracking from it, possibly involving several more jumps, eventually reaches back to v. Because v is reachable from the root and because (I1) tells us that every backtracking path from v passes through $\mathbf{find}(p_j) = p_l$, we can trace a backtracking path from v to p_l. Thus, applying the same argument to each jump, we obtain a (path with a) cycle in G, which is a contradiction. □

Lemma 3.4. *Every node is visited at most twice during the Algorithm.*

Proof. Suppose v is a sink node. It is visited once at the start of the corresponding *Backtrack*. If v is a negative atom or a $\wp^-$-node, then it will not be visited again because it has no successor. If v is a

$\otimes^-$-node, it may be revisited as a result of Case 5 or Case 8. A visit of v by Case 5 can happen only once as v has only one successor; further the successor is not a $\parr^+$-node, and so, v is never visited as a result of Case 8. Similarly a visit of v by Case 8 can only happen once and Case 5 cannot happen.

Next suppose v is not a sink node. Any run of $\mathit{Visit}(v, i)$ is called by some $\mathit{Visit}(u, i)$ either as a result of a jump from a j-labelled u to some p_l which is a successor of v (case 8), or where u is a successor of v (case 5). After that call of $\mathit{Visit}(u, i)$ completes, u has name i by Invariant (I3), and so, v will never be called by $\mathit{Visit}(u, l)$ again, for any l (even though u itself may be revisited). Since v has at most two successors, it can be visited at most twice during the Algorithm. □

Thus to summarise

Theorem 3.5 (Correctness). *Algorithm A is a linear-time algorithm for* EssNet. □

4 Extension to Proof Nets

(*In this section we let P range over linearly balanced MLL formulas, considered as proof structures.*) Suppose we have a proof structure P and a DR-switching σ for P such that the DR-graph $\mathrm{Gr}(P, \sigma)$ is connected and acyclic – this assumption is important. We give a procedure that returns an essential net which we shall write as $\mathcal{E}(P, \sigma)$. We first define an intermediate structure $\overline{P}$ which is an undirected graph obtained from the abstract syntax tree of P by drawing in the axiom links i.e. for each atom a, we add an edge joining it to $a^\perp$. We then transform $\overline{P}$ to a directed graph $\mathcal{E}(P, \sigma)$: Starting from the root of $\overline{P}$ by moving upwards, we assign a polarity to each node and perform the following edge operations:

1. assign $+$ to nodes that are reached from below
2. after crossing an axiom link, change polarity to $-$ (so that the two ends of an axiom link are given opposite polarities) then move downwards
3. assign $-$ to nodes that are reached from above.
4. The direction of traversal corresponds to the orientation of the edges incident to the polarised connectives as prescribed by the directional rules in Figure 2.
5. At a $\parr^+$-node, if σ marks it R (respectively L), we go up the right (respectively left) branch and delete the left (respectively right) edge.

Note that we perform no deletion of edges incident to a $\parr^-$-node i.e. we effectively disregard the switching at $\parr^-$-nodes. The polarity of atoms is determined solely by the rules above (so that $a^\perp$ can be assigned $+$ and hence appear as a^+ in the graph; e.g. consider $a \parr a^\perp$ with the right switch). All cases that can arise are analysed in Bellin's note [2] (see also [3]), and it is easy to see that the procedure always terminates, returning an essential net, which we write as $\mathcal{E}(P, \sigma)$. Because we work with a tree, each node is visited only once, and so, the procedure $(P, \sigma) \mapsto \mathcal{E}(P, \sigma)$ can be completed in linear time. Note that the procedure may not return an essential net, let alone a correct one, if the DR-graph $\mathrm{Gr}(P, \sigma)$ is disconnected or cyclic; e.g. consider $(a^\perp \parr b^\perp) \parr (a \parr b)$ and $(a^\perp \otimes b^\perp) \parr (a \otimes b)$.

Theorem 4.1 (Linear reducibility). *For any proof structure P*

(i) if P is a proof net then for every DR-switching σ, $\mathcal{E}(P, \sigma)$ is a correct essential net

(ii) if there exists a DR-switching σ such that $\mathcal{E}(P, \sigma)$ is a correct essential net then P is a proof net.

Algorithm B

Input: a proof structure P

1. **if** $\mathrm{Gr}(P, \sigma_R)$ disconnected or cyclic **then return no**
2. Construct $G = \mathcal{E}(P, \sigma_R)$
3. **if** G cyclic **then return no**
4. Run $\mathbf{A}'(G)$.

Procedure $\mathbf{A}'(G)$:

```
if G has form G1 ⊗+ G2
        then if G1 and G2 linearly balanced
                then A'(G1); A'(G2)
                else return no
        else A(G)
```

Figure 5: Algorithm B for ProofNet

Thus the decision problem ProofNet *is linearly reducible to* EssNet.

Proof. (Sketch) We refer the reader to [2] for a proof of (i). For (ii), suppose for some DR-switching σ for P, $\mathcal{E}(P, \sigma)$ is acyclic and every $\wp$-node satisfies Condition L, then one can effectively transform the graph to an IMLL formula A (say), such that $P = \overline{A}$ and $\mathcal{E}(A) = \mathcal{E}(P, \sigma)$. For example, take P to be $((a^{\perp} \wp b) \otimes c^{\perp}) \wp ((a \otimes b^{\perp}) \wp c)$ and σ to be the DR-switching that assigns L to every $\wp$-occurrence, then the corresponding A is $((b \multimap a) \otimes c) \ \multimap \ (b \multimap a) \otimes c$; we leave the recursive definition of the transformation as an easy exercise for the reader. By Theorem 1.3, $\vdash A$ is IMLL-provable; and so, by Proposition 1.2, P (*qua* proof structure) is a proof net. □

Combining the above with Theorem 3.5, we can construct a linear-time algorithm for ProofNet in Figure 5: we write σ_R for the DR-switching that assigns R to every occurrence of $\wp$, and $\mathbf{A}(G)$ to mean "run Algorithm **A** with input G".

5 Sequentialization

As a follow up to this work, we aim to explain the correspondence between Condition L and the notion of *justification* in game semantics (see [11]), and to show that Algorithm **B** can be extended linearly to give a "sequentialization" of a given MLL proof structure in case it is a proof net.

Acknowledgements We are grateful to Francois Lamarche and Gianluigi Bellin for useful discussions.

References

[1] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Logic*, 59:543–574, 1994.

[2] G. Bellin. Proof nets for classical MLL and linear lambda terms. A March 1992 posting to LINEAR `http://www.csl.sri.com/linear/mailing-list-traffic/www/all.html`, 1992.

[3] G. Bellin and P. J. Scott. Remarks on the π-calculus and linear logic. *Theoretical Computer Science*, 135:11–65, 1994.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

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

[6] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985.

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

[8] S. Guerrini. Correctness of multiplicative proof nets is linear. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 454–463. IEEE Computer Science Society, 1999.

[9] F. Lamarche. Proof Nets for Intuitionistic Linear Logic 1: Essential Nets. Preprint ftp-able from *Hypatia*, 1994.

[10] P. Lincoln and T. Winkler. Constant-only multiplicative linear logic is NP-complete. *Theoretical Computer Science*, 135:155–159, 1994.

[11] A. S. Murawski and C.-H. L. Ong. Exhausting Strategies, Joker Games and Full Completeness for IMLL with Unit. To appear in the ENTCS proceedings of *Category Theory and Computer Science 1999*, 34 pp., ftp-able from Ong's homepage, 1999.

A A Proof of Theorem 1.3

The soundness ($\Rightarrow$) part of the Theorem can be proved by a straightforward induction over the rules of the IMLL Sequent Calculus (less the cut rule). For the completeness part, we use a reduction argument inspired by that in [1]. We perform induction on the number of atoms for *regular* formulas which are defined by recursion over the following rules:

$$
\begin{aligned}
T &::= \underbrace{T^- \otimes \cdots \otimes T^-}_{m} \multimap \underbrace{\theta \otimes \cdots \otimes \theta}_{n} \quad | \quad \underbrace{\theta \otimes \cdots \otimes \theta}_{n} \\
T^- &::= \underbrace{T \otimes \cdots \otimes T}_{m} \multimap \theta \quad | \quad \theta
\end{aligned}
$$

where θ ranges over atoms and $m, n \geq 1$.

We can transform an irregular formula Θ to a finite set of regular ones by replacing Θ by Θ_1, Θ_2 repeatedly, using the following reduction rules:

Θ	Θ_1	Θ_2
$C^+[A \otimes (B \multimap C)]$	$C^+[(A \multimap B) \multimap C]$	$C^+[B \multimap A \otimes C]$
$C^+[(B \multimap C) \otimes A]$	$C^+[(A \multimap B) \multimap C]$	$C^+[B \multimap A \otimes C]$
$C^-[A \multimap B \otimes C]$	$C^-[(A \multimap B) \otimes C]$	$C^-[(A \multimap C) \otimes B]$
$C[A \multimap (B \multimap C)]$	$C[A \otimes B \multimap C]$	

where C is a one-holed *type* context, C^+ is a context whose hole occurs in a positive position, and C^- is a context whose hole occurs in a negative position. First we note the following (for a proof see e.g. [1]):

Proposition A.1. *For types Θ, Θ_1 and Θ_2 as given in the first three reduction rules, if $\vdash \Theta_1$ and $\vdash \Theta_2$ then $\vdash \Theta$.* $\square$

The reduction process is strongly normalising. A measure on types can be defined as follows:

$$\begin{array}{rclcl} \mu(\theta) & = & 2 & & \\ \mu(D \otimes^+ E) & = & \mu(D \multimap^- E) & = & \mu(D) \times \mu(E) \\ \mu(D \otimes^- E) & = & \mu(D \multimap^+ E) & = & \mu(D) + \mu(E) \end{array}$$

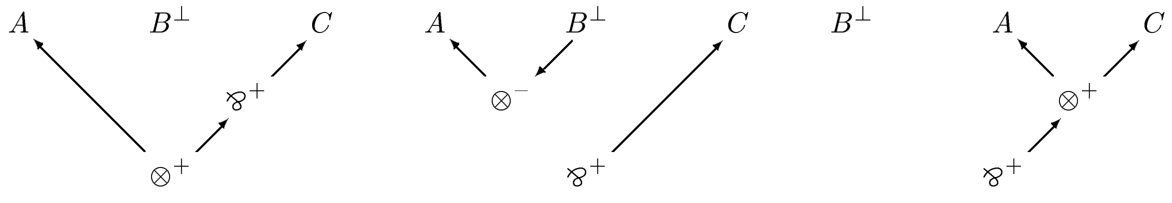
The measure is strictly decreasing for the first three reduction rules; it is invariant for the last but note that the number of occurrences of $\multimap$ strictly decreases in that case. Plainly the normal forms of the reduction are regular by design.

Thus the reduction terminates with a finite set of regular formulas. In fact, each of these formulas is guaranteed to induce a correct essential net, which we know because of the following lemma.

Lemma A.2. *For Θ, Θ_1 and Θ_2 as given in the first reduction rules, if $\mathcal{E}(\Theta)$ is correct, then so is $\mathcal{E}(\Theta_1)$ and $\mathcal{E}(\Theta_2)$.*

Proof. We show that the correctness criteria are preserved by a case-by-case analysis of the transformed graph. We draw the subgraph corresponding to Θ, Θ_1, and Θ_2.

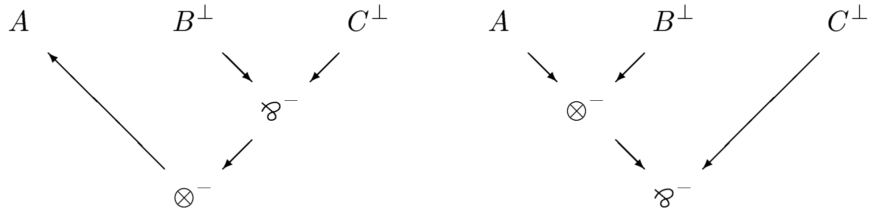
(i), (ii): The two cases are symmetric so we consider the first one only.



If there were a cycle in $\mathcal{E}(\Theta_1)$, it would have to involve the $\otimes^-$ node as otherwise it would be in $\mathcal{E}(\Theta)$. However, this would mean that Condition L for $\mathcal{E}(\Theta)$ is violated. Acyclicity of $\mathcal{E}(\Theta_2)$ follows from acyclicity of $\mathcal{E}(\Theta)$.

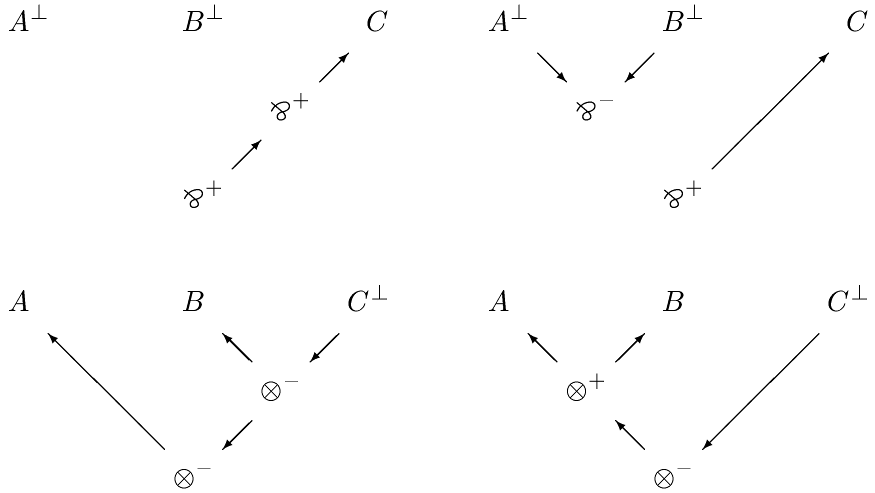
To see that $\mathcal{E}(\Theta_1)$ satisfies Condition L, note that backtracking paths from $B^\perp$ and C are unaffected by the transformation and for A they eventually reach the $⅋^+$ node (by Condition L for $\mathcal{E}(\Theta)$ and the fact that it must lead through $B^\perp$) as in $\mathcal{E}(\Theta)$ they reach $\otimes^+$. Thus it is impossible to bypass any $\parr^+$ nodes, and so, Condition L for $\mathcal{E}(\Theta_1)$ can be derived from Condition L for $\mathcal{E}(\Theta)$. It is straightforward to see that for $\mathcal{E}(\Theta_2)$ Condition L holds.

(iii): Because of symmetry, it suffices to consider just one of Θ_1 and Θ_2. We draw it on the right.



Note that the reachability relation between A, $B^\perp$ and $C^\perp$ of the graph on the right is included in that of the left graph which accounts for the preservation of acyclicity and Condition L.

(iv): There are two cases here depending on the polarity of the context. The pictures speak for themselves.



□

Thus we can prove the completeness part of the theorem by induction for the special case where Θ is regular. We can then extend the result to the general case by appealing to Proposition A.1.

The following lemmas deal with the induction step in the regular case.

Lemma A.3 (Splitting). *Suppose for each i, $S_i = T_i \multimap a_i$ or $S_i = a_i$ where a_i is an atom and $\mathcal{E}(S_1 \otimes \cdots \otimes S_n \multimap M_1 \otimes M_2)$ is correct. Then there exists a (unique) partition of $\{1, \cdots, n\}$ into X_1 and X_2 such that $\mathcal{E}(\bigotimes_{x \in X_i} T_x \multimap M_i)$ is correct.*

Proof. Let us construct an undirected graph with vertices $T_1, \cdots, T_n, M_1, M_2$. There is an edge between T_i and some other vertex V if V contains the positive occurrence of a_i. Thus each T_i is the end of precisely one link. Now take X_i to be the set of indices of T_j's in the same component as M_i. By previous remarks X_1 and X_2 are disjoint. By Remark 1.4 each $a_i^\perp$ can be reached from the root in $\mathcal{E}(S_1 \otimes \cdots \otimes S_n \multimap M_1 \otimes M_2)$. Since the only way of entering (the part corresponding to) S_i in the essential net is through the axiom link between $a_i^\perp$ and a_i, if a_i can be reached via M_j, T_i is in the same component as M_j. Moreover, each path in the essential net that visits M_j can only pass through the nodes corresponding to the T_i's lying in the component containing M_j. Therefore the partition yields two correct essential nets. $\square$

Lemma A.4. *Suppose the essential net of a regular formula of the form $T_1^- \otimes \cdots \otimes T_n^- \multimap a$ is correct. Then for some T' and some i, we have $T_i^- = T' \multimap a$ and the essential net for*

$$T_1^- \otimes \cdots \otimes T_{i-1}^- \otimes T_{i+1}^- \otimes \cdots \otimes T_n^- \multimap T'$$

is again correct.

Proof. Recall that each T_i^- is of shape $T^i \multimap x$ and the only way of entering the part corresponding to T_i^- in the essential net is via the axiom link of $x^\perp$. The first node reachable from the root is a so the next one will be $a^\perp$. As it must be part of some T_i^-, that T_i^- is of shape $T' \multimap a$. The transformed essential net must be correct as its structure is inherited from the old one. $\square$

We can now give the inductive argument. If Θ has only two atoms, we have $\Theta = a \multimap a$, so the corresponding sequent is IMLL-provable. The lemmas above enable us to reduce the size of regular formulas. Note that T' has the shape $T \otimes \cdots \otimes T$ and we can use Lemma A.3 to consider formulas of the shape $T^- \otimes \cdots \otimes T^- \multimap T$. Then T has the shape $T^- \otimes \cdots \otimes T^- \multimap \theta \otimes \cdots \otimes \theta$ and by using the last regular transform, we get a truly regular formula.