A New Linear Time Correctness Condition for Multiplicative Linear Logic

Satoshi Matsuoka
National Institute of Advanced Industrial Science and Technology (AIST),
1-1-1 Umezono, Tsukuba, Ibaraki, 305-8565 Japan
matsuoka@ni.aist.go.jp

Abstract
In this paper, we give a new linear time correctness condition for proof nets of Multiplicative Linear Logic without units. Our approach is based on a rewriting system over trees. We have only three rewrite rules. Compared with previous linear time correctness conditions, our system is surprisingly simple and intuitively appealing.

1 Introduction
More than three decades ago, J.Y. Girard introduced the notion of proof nets of unit free Multiplicative Linear Logic (for short, MLL)\cite{girard87}. It is a parallel syntax for MLL proofs, removing redundancy of sequent calculus proofs. In \cite{girard87}, he introduced MLL proof structures, which are graphs whose nodes are labeled by MLL formulas and then defined MLL proof nets as sequentializable MLL proof structures. Moreover he introduced a topological property called the long trip condition for MLL proof structures and showed that an MLL proof structure is an MLL proof net if and only if it satisfies the long trip condition. Such a characterization is called a correctness condition for MLL proof nets. Since then many other correctness conditions have been given for MLL and its variants or extensions by many authors.

Complexity questions about correctness conditions come up naturally. The first linear time correctness condition for MLL is given in \cite{decazeauc02}, which is based on contractability condition \cite{decazeauc02}. Another linear time correctness conditions are given in \cite{decazeauc02}, which are based on that of essential nets, which are an intuitionistic variant of MLL proof nets. Moreover de Naurois and Mogbil introduce a correctness condition for MLL and their extensions based on topological conditions of arbitrarily selected one DR-graph \cite{decazeauc02} and showed that they are NL-complete \cite{decazeauc02}.

In this paper we introduce a new linear time correctness condition for MLL. It is based on that of \cite{decazeauc02}. Although de Naurois and Mogbil showed...
their correctness condition is NL-complete, its linear time complexity cannot be derived from their presentation in [4] directly. In order to establish the linear time complexity, we define a rewriting system over DR-trees, which are basically special cases of DR-graphs. The rewriting system have only three rewrite rules, which is remarkably simple. In the rewriting system, an active node flows in a DR-tree, reducing nodes by the rewriting rules. However, the rewriting system may lead to quadratic time termination at the worse case. In order to fix the situation, we introduce more sophisticated data structures and a rewriting strategy. Thanks to them, we can establish the linear time correctness condition. Compared with [7] and [9], our correctness condition is surprisingly simple and intuitively appealing. While the correctness condition in [7] has to use non-local jump rule called the new rule, all three rewriting rules in our system are strictly local. In addition, all correctness conditions in [9] are rather complex since there must be several different passes in the algorithm or they must be combined in one pass. Our rewriting system has just three rewrite rules.

2 Multiplicative Linear Logic, Proof Structures and Proof Nets

2.1 Multiplicative Linear Logic

We introduce the system of Multiplicative Linear Logic (for short MLL). We define MLL formulas, which are denoted by $F, G, H, \ldots$, by the following grammar:

$$F ::= p \mid p^\bot \mid F \otimes G \mid F \parr G$$

The negation of $F$, which is denoted by $F^\bot$ is defined as follows:

$$(p)^\bot = p^\bot, \quad (F \otimes G)^\bot = G^\bot \otimes F^\bot, \quad (F \parr G)^\bot = G^\bot \parr F^\bot$$

We denote multisets of MLL formulas by $\Lambda, \Lambda_1, \Lambda_2, \ldots$. An MLL sequent is a multiset of MLL formulas $\Lambda$. We write an MLL sequent $\Lambda$ as $\vdash \Lambda$. The inference rules of MLL are as follows:

| ID | $\vdash F^\bot, F$ |
| silence | $\vdash A_1, F \vdash A_2, G$ | $\vdash F, G$ |
| silence | $\vdash A_1, F \vdash A_2, F^\bot$ |

We omit the cut rule that has the form

$$\text{Cut} \quad \vdash A_1, F \quad \vdash A_2, F^\bot \quad \vdash A_1, A_2$$

because it can be identified as the $\otimes$-rule for our purpose.
2.2 MLL Proof Nets

Next we introduce MLL proof nets. Figure 1 shows the MLL links we use. Each MLL link has a few MLL formulas. Such an MLL formula is a conclusion or a premise of the MLL link, which is specified as follows:

1. In an ID-link, each of $F$ and $F^\perp$ is called a conclusion of the link;
2. In an $\otimes$-link, each of $F$ and $G$ is called a premise of the link and $F \otimes G$ is called a conclusion of the link;
3. In an $\forall$-link, each of $F$ and $G$ is called a premise of the link and $F \forall G$ is called a conclusion of the link.

Figure 1: MLL Links

An MLL proof structure $\Theta$ is a set of MLL links that satisfies the following conditions:

1. For each link $L$ in $\Theta$, each conclusion of $L$ can be a premise of at most one link other than $L$ in $\Theta$;
2. For each link $L$ in $\Theta$, each premise of $L$ must be a conclusion of exactly one link other than $L$ in $\Theta$.

A formula occurrence $F$ in an MLL proof structure $\Theta$ is a conclusion of $\Theta$ if $F$ is not a premise of any link in $\Theta$.

An MLL proof net is an MLL proof structure that is constructed by the rules in Figure 2. Note that each rule in Figure 2 has the corresponding inference rule in the MLL sequent calculus. Any MLL proof structure is not necessarily an MLL proof net.

Next we introduce a characterization of MLL proof nets using the notion of DR-switchings. A DR-switching $S$ for an MLL proof structure $\Theta$ is a function from the set of $\forall$-links in $\Theta$ to $\{0, 1\}$. The DR-graph $S(\Theta)$ for $\Theta$ and $S$ is defined by the rules of Figure 3. Then the following characterization holds.

**Theorem 2.1** ([3]) An MLL proof structure $\Theta$ is an MLL proof net if and only if for any DR-switching $S$ for $\Theta$, the DR-graph $S(\Theta)$ is acyclic and connected.
Figure 2: Definition of MLL Proof Nets

1. If \( F \) occurs in \( \Theta \) then \( F \) \( \perp \) is an edge of \( S(\Theta) \)

2. If \( F \) and \( G \) occur in \( \Theta \) then \( F \) \( G \) \( F \odot G \) are two edges of \( S(\Theta) \)

3. If \( F \) and \( G \) occur in \( \Theta \) then \( F \) \( G \) \( \eta G \) is an edge of \( S(\Theta) \)

4. If \( F \) and \( G \) occur in \( \Theta \) then \( F \) \( G \) \( \eta G \) is an edge of \( S(\Theta) \)

Figure 3: Definition of DR graphs
2.3 de Naurois and Mogbil’s correctness condition

In this section we review de Naurois and Mogbil’s correctness condition [4], on which our linear time condition is based.

Definition 2.1 A DR-switching $S$ for an MLL proof structure $\Theta$ is extreme left if for each $\&$-link $L$ in $\Theta$, $S$ always chooses the left premise in $L$. We denote the DR-switching by $S_{\forall \ell}$.

In the following we only consider the extreme left switching. We have no loss of generality under the assumption.

Definition 2.2 Let $\Theta$ be an MLL proof structure such that the DR-graph $S_{\forall \ell}(\Theta)$ is a tree. Let $L$ be a $\&$-link, and $n_L$, $n_L^\ell$, and $n_L^r$ be nodes in $S_{\forall \ell}(\Theta)$ induced by $L$, left, and right premises of $L$ respectively. We say that $L$ is consistent in $S_{\forall \ell}(\Theta)$ if the unique path $\theta$ from $n_L^\ell$ to $n_L^r$ in $S_{\forall \ell}(\Theta)$ does not contain $n_L$.

Definition 2.3 Let $\Theta$ be an MLL proof structure such that the DR-graph $S_{\forall \ell}(\Theta)$ is a tree and each $\&$-link in $\Theta$ is consistent in $S_{\forall \ell}(\Theta)$. Then we define a directed graph $G(S_{\forall \ell}(\Theta)) = (V, E)$ as follows:

- $V = \{n_L | L$ is a $\&$-link in $\Theta\}$

- Let $L_1, L_2$ be different $\&$-links in $\Theta$. The directed edge $(n_{L_1}^\ell, n_{L_2}^r)$ in $E$ if the unique path from $n_{L_1}^\ell$ to $n_{L_2}^r$ in $S_{\forall \ell}(\Theta)$ contains the node $n_{L_1}$.

Theorem 2.2 ([4]) An MLL proof structure $\Theta$ is an MLL proof net iff

1. The DR-graph $S_{\forall \ell}(\Theta)$ is a tree.

2. Each $\&$-link in $\Theta$ is consistent in $S_{\forall \ell}(\Theta)$. ($\&$-link consistency)

3. The directed graph $G(S_{\forall \ell}(\Theta))$ is acyclic. (directed acyclicity)

3 The Rewriting System over deNM-Trees

In this section we introduce our rewriting system. Then we give our correctness condition based on the system and show that it is a characterization of MLL proof nets.

3.1 deNM-trees

First we define deNM-trees. In the following we fix an MLL proof structure $\Theta$ such that the DR-graph $S_{\forall \ell}(\Theta)$ is a tree.

Definition 3.1 (deNM-trees) A $\&$-label is $\ell_L$ or $r_L$ where $L$ is a $\&$-link $L$ in $\Theta$. 
• A label set is a finite set whose elements are $\mathcal{H}$-labels.

• A labeled node is a node for a deNM-tree that is labeled by a label set. The degree $t$ of a labeled node is at most the number of nodes of the deNM-tree containing itself. See Figure 4.

• A $\mathcal{H}$-node is a node for a deNM-tree that is labeled by a $\mathcal{H}$-link $L$ in $\Theta$. The degree of a $\mathcal{H}$-node is 1 or 2. See Figure 5. As shown symbolically, we distinguish the port above of a $\mathcal{H}$-node from the port below.

• A deNM-tree is a finite tree consisting of labeled nodes and $\mathcal{H}$-nodes. In a similar manner to Definition 2.2, we can define $\mathcal{H}$-consistency over deNM-trees. In addition, in a similar manner to Definition 2.3 a directed graph from a deNM-tree and its acyclicity can be defined.

\[\text{labeled-node}\]

\[
\begin{array}{c}
\ldots \\
\_ \_ \_ \_ \_ \_ \_ \\
S
\end{array}
\]

Figure 4: Labeled nodes

\[\text{\mathcal{H}-node (degree 1)} \quad \text{\mathcal{H}-node (degree 2)}\]

\[\begin{array}{c}
\_ \\
\_ \\
L
\end{array} \quad \begin{array}{c}
\_ \\
\_ \\
L
\end{array}\]

Figure 5: $\mathcal{H}$-nodes

Next we give a translation from $\Theta$ to a deNM-tree.

**Definition 3.2** We define a deNM-tree $T(\Theta)$ from $\Theta$ such that the DR-graph $S_{\mathcal{H}}(\Theta)$ is a tree. If $\Theta$ consists of exactly one ID-link, then $T(\Theta)$ is a tree that consists of exactly one 0-degree node labeled by $\emptyset$. Otherwise, for each link $L$ in $\Theta$ we specify a subtree $T_L$ in $T(\Theta)$ corresponding to $L$ as follows:

• The case where $L$ is ID-link:
1. The case where one conclusion of \( L \) is a right premise \( F \) of a \( \otimes \)-link \( L' \) or a conclusion \( F \) of \( \Theta \): Then \( T_L \) consists of exactly one labeled node \( n_L \) with degree 1 that is connected to the translation of the other conclusion \( F' \) of \( L \) (more precisely, \( T_L \) is connected to the translation of the link whose left or right premise is \( F' \)). Without loss of generality, we can assume that \( F' \) is not a premise of \( \otimes \)-link because otherwise, we can easily see that \( \Theta \) is not an MLL proof net (in this case we define \( T(\Theta) \) to be undefined). Then if the conclusion of \( L \) is a right premise of \( L' \), then the labeled set of \( n_L \) is \( \{r_{L'}\} \). Otherwise, that of \( n_L \) is empty. See Figure 6.

![Figure 6: ID-link (Case 1)](image_url)

2. Otherwise: In this case without loss of generality we can assume that one of the conclusions of \( L \) is not a premise of a \( \otimes \)-link because when both conclusions of \( L \) are a premise of a \( \otimes \)-link, we can easily see that \( \Theta \) is not an MLL proof net (in this case we define \( T(\Theta) \) to be undefined). Then \( T_L \) consists of exactly one labeled node \( n_L \) with degree 2. If one of the conclusions of \( L \) is a left premise of a \( \otimes \)-link \( L' \), then the labeled set for \( n_L \) is \( \{\ell_{L'}\} \). Otherwise the labeled set for \( n_L \) is \( \emptyset \). See Figure 7.

- The case where \( L \) is \( \otimes \)-link:

1. The case where the conclusion of \( L \) is a conclusion of \( \Theta \) or a right premise of a \( \otimes \)-link \( L' \): In this case \( T_L \) consists of exactly one labeled node \( n_L \) with degree 2 that is connected to trees translated from both premises of \( L \). If the conclusion of \( L \) is a right premise of \( L' \), then the labeled set for \( n_L \) is \( \{r_{L'}\} \). Otherwise, the labeled set for \( n_L \) is \( \emptyset \). See Figure 8.
Figure 7: ID-link (Case 2)

Figure 8: ⊗-link (Case 1)
2. Otherwise: In this case $T_L$ consists of exactly one labeled node $n_L$ with degree 3. If the conclusion of $L$ is a left premise of a $\otimes$-link $L'$, then the labeled set for $n_L$ is $\{\ell_{L'}\}$. Otherwise, the labeled set for $n_L$ is $\emptyset$. See Figure 9.

![Figure 9: $\otimes$-link (Case 2)](image)

- The case where $L$ is $\otimes$-link:
  1. The case where the conclusion of $L$ is a right premise of a $\otimes$-link $L'$: In this case $T_L$ consists of one labeled node $n_1$ with degree 1 labeled by $\{r_{L'}\}$ and one $\otimes$-node $n_L$ labeled by $L$ with degree 2 such that $n_1$ and $n_L$ is connected. The node $n_L$ is connected to the tree translated from the left premise of $L$. See Figure 10.

![Figure 10: $\otimes$-link (Case 1)](image)

2. The case where the conclusion of $L$ is a left premise of a $\otimes$-link $L'$: In this case $T_L$ consists of one labeled node $n_2$ with degree
2 labeled by \{\ell_L\} and one \&-node \(n_L\) labeled by \(L\) with degree 2 such that \(n_2\) and \(n_L\) is connected. While \(n_L\) is connected to the tree translated from the left premise of \(L\), \(n_2\) is connected to the tree translated from the conclusion of \(L\). See Figure 11.

![Figure 11: \&-link (Case 2)](image)

3. Otherwise: In this case \(T_L\) consists of exactly one \& node \(n_L\) labeled by \(L\). If \(L\) is a conclusion of \(\Theta\), then the degree of \(n_L\) is 1. Otherwise, the degree of \(n_L\) is 2. See Figure 12.

![Figure 12: \&-link (Case 3)](image)

Then \(T(\Theta)\) is the tree obtained by connecting these subtrees \(T_L\).

If \(T(\Theta)\) is defined, then we can easily see that \(T(\Theta)\) is a deNM-tree because we assume that \(S_{\forall\ell}(\Theta)\) is a tree.

**Proposition 3.1**  
(a) Let \(\Theta\) be an MLL proof net. Then \(T(\Theta)\) satisfies \&-consistency and directed acyclicity for deNM-trees.

(b) \(T\) be an deNM-tree that satisfies \&-consistency and directed acyclicity. Let \(T'\) be an deNM-tree obtained from \(T\) by choosing one active node
applying one of three rewrite rules to \( n \). Then \( T' \) satisfies \( \mathcal{E} \)-consistency and directed acyclicity for deNM-trees.

**Proof:**

(a) It is obvious from Theorem 2.2.

(b) Each rewrite rule preserves \( \mathcal{E} \)-consistency and directed acyclicity.

3.2 The Rewriting System over deNM-Trees

Next we introduce our rewriting system over deNM-trees. In the rewriting system we must specify exactly one node in a deNM-tree that is about to be rewritten, which we call the active node in the deNM-tree. The active node must be a labeled node. Our rewriting system has only three rewrite rules.

- The rewrite rule of Figure 13 is called \( \mathcal{E} \)-elimination: if the active node \( n \) is connected to a \( \mathcal{E} \)-node \( n_L \) labeled by \( L \) through the port above and the label set \( S \) of \( n \) contains labels \( \ell_L \) and \( r_L \), then \( n_L \) is eliminated.

![Figure 13: \( \mathcal{E} \)-elimination rule](image)

- The rewrite rule of Figure 14 is called union: If the active node is connected to a labeled node, then these two nodes are merged. The label set of the resulting node is the union of them of merged two nodes.

![Figure 14: Union rule](image)

- The rewrite rule called local jump of Figure 15 does not change any nodes: It just change the current active node. Note that in this rewrite rule, the active node \( n \) is connected to a \( \mathcal{E} \)-node through the port below.
Figure 15: Local jump rule

We denote the rewriting system consisting of the three rewrite rules above by $\mathcal{R}$.

Let $\Theta$ be an MLL proof structure and $\mathbb{L}_\Theta = \{L_1, \ldots, L_m\}$ be the set of all $\&$-links in $\Theta$. Then we define the full label set $S_{\text{full}}$ to be

$$S_{\text{full}} = \{\ell_L \mid L \in \mathbb{L}_\Theta\} \cup \{r_L \mid L \in \mathbb{L}_\Theta\}$$

**Definition 3.3** Algorithm $A$ is defined as follows:

*Input:* an MLL proof structure $\Theta$

*Output:* *yes* or *no*.

1. **If** DR-graph $S_{\text{def}}(\Theta)$ is not a tree, then the output is *no*. Otherwise go to 2.
2. **If** the deNM-tree $T(\Theta)$ is not defined, then the output is *no*. Otherwise go to 3.
3. An labeled node $n$ in $T(\Theta)$ is selected arbitrarily.
4. Start rewriting with $T(\Theta)$ and the active node $n$ using three rewrite rules above.
5. **If** the local jump rule is tried to be applied to a $\&$-link to which the local jump rule was applied before, then the output is *no*.
6. **When** any of three rewrite rules cannot be applied to the current deNM-tree $T'$, if $T'$ consists of exactly one node labeled by $S_{\text{full}}$ with degree 0, then the output is *yes*. Otherwise, the output is *no*.

**Proposition 3.2** Algorithm $A$ always terminates.

**Proof:** Algorithm $A$ cannot be applied the jump rule to a $\&$-link more than one time. Both of the other two rules reduce the number of nodes. \(\square\)

**Lemma 3.1** If Algorithm $A$ terminates in step 5, then $\Theta$ is not an MLL proof net.
Proof: In this case, \( T(\Theta) \) must reduce to a deNM-tree with configuration shown in Figure 16. Then if \( \Theta \) does not violate the second condition of Theorem 2.2 then the configuration must extend to the configuration shown in Figure 17. But it is not a tree anymore. Therefore \( \Theta \) is not an MLL proof net. \( \square \)

\[ \text{Figure 16: Configuration for Termination at step 5} \]

\[ \text{Figure 17: But not a deNM-tree anymore} \]

**Theorem 3.1** Let \( \Theta \) be an MLL proof structure. Then \( \Theta \) is an MLL proof net if and only if Algorithm A with input \( \Theta \) outputs yes.

Proof:
• Only-if-part: We suppose that Algorithm A with input Θ outputs no. By the characterization of Theorem 2.1 Algorithm A reaches step 5. Then application of the jump rule to a \( \varphi \)-link twice means that Θ is not an MLL proof net by Lemma 3.1. This is a contradiction. So Algorithm A applies the jump rule to each \( \varphi \)-link at most one time. Algorithm A terminates (Proposition 3.2). We suppose that the deNM-tree at the termination is not one node tree. Since the union and \( \varphi \)-elimination rules preserve \( \varphi \)-consistency and \( \varphi \)-link directed acyclicity (Proposition 3.1), \( T(\Theta) \) must violate one of the second and third conditions in Theorem 2.2. Since Θ is an MLL proof net, this is a contradiction. This means that it must terminates with exactly one node tree with degree 0. Moreover, the one node must be labeled by \( S_{\text{full}} \).

• If-part: We suppose that Algorithm A with input Θ outputs yes. Then Θ automatically satisfies the first condition of Theorem 2.2. We suppose that there is an inconsistent \( \varphi \)-link \( L \) in \( S_{\forall\ell}(\Theta) \). Then our rewriting system cannot be reduced \( T(\Theta) \) to one node tree, because we cannot apply the \( \varphi \)-elimination rule to the \( \varphi \)-node \( n_L \). So Θ satisfies the second condition of Theorem 2.2. We suppose that the directed graph \( G(S_{\forall\ell}(\Theta)) \) has a cycle. Then our rewriting system cannot be reduced \( T(\Theta) \) to one node tree, because we cannot apply the \( \varphi \)-elimination rule to the \( \varphi \)-links which are contained in the cycle. Hence Θ satisfies the third condition of Theorem 2.2. Therefore Θ must be an MLL proof net. □

3.3 Examples

We show three examples in this section. Figure 18 shows an MLL proof net Θ₁, where the symbol ⊙ means a \( \varphi \)-link occurrence. This figure has been generated using the Proof Net Calculator [8]. It is translated to the deNM-tree \( T(\Theta_1) \) shown in Figure 19. When you choose any labeled node as the starting active node, you must finally reach to one labeled node with degree 0 labeled by the full label set

\[
S_{\text{full}} = \{\ell_1, r_1, \ell_2, r_2, \ell_3, r_3, \ell_4, r_4\}
\]

using our three rewrite rules.

Figure 20 shows an MLL proof structure Θ₂ that is not an MLL proof net. It is translated to the deNM-tree \( T(\Theta_2) \) shown in Figure 21. When you choose any labeled node as the starting active node, you cannot reach to one labeled node with degree 0 labeled by the full label set

\[
S_{\text{full}} = \{\ell_1, r_1, \ell_2, r_2, \ell_3, r_3, \ell_4, r_4\}
\]

using our three rewrite rules.
Figure 18: MLL Proof Net $\Theta_1$

Figure 19: $T(\Theta_1)$
Figure 20: MLL Proof Structure $\Theta_2$, but not MLL Proof Net

Figure 21: $T(\Theta_2)$
Figure 22 shows an MLL proof structure $\Theta_3$ that is not an MLL proof net. It is translated to the deNM-tree $T(\Theta_3)$ shown in Figure 23. When we choose the node labeled by $\{r_2\}$ as the starting rule, the first rewrite rule may be the local jump rule for $\&$-link 1. Then the node labeled by $\{r_2\}$ becomes active. After two applications of the union rule, the local jump rule for $\&$-link 1 must be tried to be applied again. Then step 5 in Algorithm A outputs no.

Figure 22: MLL Proof Structure $\Theta_3$, but not MLL Proof Net

Figure 23: $T(\Theta_3)$
3.4 Linear Time Correctness Condition

Although our rewriting system $R$ is surprisingly simple, it cannot establish linear time termination, because nodes in a deNM-tree $T$ may have degrees depending on the number of nodes of $T$ and take quadratic time. For example, reduction of the deNM-tree shown in Figure 24 to one node tree may take quadratic time $O(k^2)$ in $R$ because before each application of the union rule it may try to apply the $\&$-elimination rule to the active node and $\&$-node $i$ $(1 \leq i \leq k)$. In order to establish linear time termination based on our rewriting system, we must fix a reduction strategy.

Let $\Theta$ be an MLL proof structure and $T$ be a deNM tree during reduction based on our reduction strategy, starting from $T(\Theta)$. We assume that each node $n$ in $T$ has the following data structures:

- The list $L_{\text{down}}$ of $\&$-nodes that connect to $n$ from the port below.
- The list $L_{\text{labeled}}$ of labeled-nodes that connects to $n$.
- The list $L_{\text{right}}$ of right labels $r_L$ on $n$ that have not been tried for $\&$-elimination yet. Initially, if the label node is labeled by $r_L$, then $L_{\text{right}} = r_L$. Otherwise, $L_{\text{right}}$ is empty.
- The set $S_{\text{right}}$ of right labels $r_L$ on $n$ that have already been tried for $\&$-elimination, where the set is in the sense of the disjoint set-union data structure $[1]$. Initially, $S_{\text{right}}$ is always empty. We call $S_{\text{right}}$ the right label set for $n$.
- The list $L_{\text{up}}$ of $\&$-nodes that connect to $n$ from the port above that have not been tried for $\&$-elimination yet.
- The set $S_{\text{up}}$ of $\&$-nodes that connect to $n$ from the port above and have already been tried for $\&$-elimination, where the set is in the sense of the disjoint set-union data structure $[1]$. We call $S_{\text{up}}$ the up port set for $n$. Initially, $S_{\text{up}}$ is always empty.

In the initial stage, we can associate these data structures to each node in $T(\Theta)$ in linear time.

Figure 24: A deNM-Tree that may needs quadratic computation at the worst case scenario
Let the current active node in $T$ be $n_{act}$. Then we define our reduction strategy as follows:

1. First if $L_{down}$ for $n_{act}$ is not empty, then it apply the local jump rule to $n_{act}$ and the first element of $L_{down}$. After the application, the first element is deleted from $L_{down}$.

2. Second in the case where $L_{down}$ for $n_{active}$ is empty and $L_{labeled}$ for $n_{act}$ is not empty, if the first element $n'$ of $L_{labeled}$ does not denote itself, i.e., $n_{act}$, then it apply the union rule to $n_{act}$ and $n'$. After the application, the data structures for two nodes $n_{act}$ and $n'$ are merged in such a way that $n'$ is deleted from new $L_{labeled}$. These merges can be done in constant time. Otherwise, i.e., if the first element of $L_{labeled}$ denote itself, then the element is deleted from $L_{labeled}$ and return to the beginning of this step.

3. Third in the case where both $L_{down}$ and $L_{labeled}$ for $n_{active}$ are empty, if $n_{active}$ is one node tree, then the output is yes. Otherwise, both $L_{right}$ and $L_{up}$ are empty, then the output is no.
   - if $L_{right}$ is not empty, let the first element of $L_{right}$ be $r_{L}$. If $S_{up}$ includes $\varphi$-link $L$, then we apply the $\varphi$-elimination rule to $n_{act}$ and $L$. The data structures for two nodes $n_{act}$ and $L$ are merged in such a way that $r_{L}$ is deleted from new $L_{right}$. These merges can be done in constant time. Otherwise, i.e., if $S_{up}$ includes $\varphi$-link $L$, then $r_{L}$ is deleted from new $L_{right}$ and put in $S_{right}$. These operations can be done in constant time.
   - if $L_{up}$ is not empty, let the first element of $L_{up}$ be $\varphi$-link $L$. If $S_{right}$ includes $r_{L}$, then then we apply the $\varphi$-elimination rule to $n_{act}$ and $L$. The data structures for two nodes $n_{act}$ and $L$ are merged in such a way that $L$ is deleted from new $L_{up}$. These merges can be done in constant time. Otherwise, i.e., if $S_{right}$ does not includes $r_{L}$, then $L$ is deleted from new $L_{up}$ and put in $S_{up}$. These operations can be done in constant time.

**Definition 3.4** We call the modified Algorithm A with the data structures and the strategy above Algorithm B.

**Remark 1**

- When the local jump rule is applied, the first element of $L_{down}$ is deleted. Therefore unlike the rewriting system $R$, we do not need the second application check of the local jump rule to the same $\varphi$-link. But in order to detect a cycle in a DR-graph as soon as possible, this check may be included.

- In order to establish linear time termination, we cannot maintain the set of $\varphi$-nodes that connect to the active node from the port above
as a queue data structure. For example, if the active node in deNM- 
tree shown in Figure 25 maintain the information as the list 1, 2, . . . , k, 
then we must scan the queue and delete one element at each \( \forall \)-elimination,
so that the reduction to one node tree takes quadratic time at the worst case.

- Instead of the queue data structure, we use the disjoint set-union data
structure [1]. Thanks to the data structure, the amortized cost is linear.

- We do not delete the eliminated \( \forall \)-link from \( S_{up} \) nor the right
label \( r_L \) from \( S_{right} \), because the cost may be linear, implying quadratic
time termination at the worst case and the deletion is not necessary.

Figure 25: A deNM-Tree that needs quadratic computation at the worst case scenario

**Theorem 3.2** Let \( \Theta \) be an MLL proof structure. There is a random access
machine that simulate Algorithm B with input \( \Theta \) in \( O(n) \) time, where \( n \) is
the number of the links in \( \Theta \).

**Proof:** The tree check for \( S_{\forall \ell}(\Theta) \) can be computed in \( O(n) \) by using
breadth-first or depth-first search. If \( S_{\forall \ell}(\Theta) \) is a tree, then \( T(\Theta) \) can also
be obtained and an arbitrary labeled node in \( T(\Theta) \) be found in \( O(n) \). In
the tree rewriting process, each node can be visited at most twice. The
total number of applications of three rewrite rules is \( O(n) \). The only
nontrivial part is management of right label sets \( S_{\text{right}} \) and that of up port
sets \( S_{up} \). The union operation in the union rule and the query (find)
operations in the \( \forall \)-elimination and local jump rules are a typical instance
of disjoint-set union-find operations [1]. That is also applies to the union
and query (find) operations for up port sets in our reduction strategy. At
general case, the amortized cost of these operations is superlinear.
However, if the underlying structure is a tree known in advance and the union operation is only performed between one node and its parent, then the amortized cost is linear in the total number of both operations in the random access machine model \[5\]. Luckily this applies to our case. Therefore our claim holds. \[\square\]

**Example 3.1** We consider the MLL proof net \(\Theta_4\) shown in Figure 26. The proof net \(\Theta\) is translated to the deNM-tree \(T(\Theta)\) shown in Figure 27. The deNM-tree \(T(\Theta_4)\) needs \(O(k^2)\) computations in \(\mathcal{R}\) when starting from the node labeled by \(\{\ell_{2k}\}\) at the worst scenario. Let us start rewriting based on our reduction strategy. In the starting active node has \(L_{\text{down}} = 1\) and \(L_{\text{up}} = 2k\). On the other hand, \(L_{\text{labeled}}, L_{\text{right}}, S_{\text{right}}, L_{\text{up}},\) and \(S_{\text{up}}\) are all empty. After several rewriting steps, we reach the deNM tree shown in Figure 28. Then for example, the active node has \(L_{\text{up}} = 1, 2, \ldots, k - 1, k\) and \(L_{\text{right}} = r_{2k}, r_k\). On the other hand, \(L_{\text{down}}, L_{\text{labeled}}, S_{\text{right}},\) and \(S_{\text{up}}\) are all empty. Our reduction strategy have some choices about which passive node is chosen for application of the union rule. But any choice leads to linear time termination. To each element in \(L_{\text{up}}\), application of the \(\otimes\)-elimination rule is tried. But \(S_{\text{right}}\) is empty, all attempts fail. Then \(S_{\text{up}}\) becomes \(\{1, 2, \ldots, k - 1, k\}\). Next to each element in \(L_{\text{right}}\) application of the \(\otimes\)-elimination rule is tried. Then \(r_{2k}\) fails, but \(r_k\) succeeds. Then we get the deNM-tree \(T_1\) shown in Figure 28. In the new active node, \(S_{\text{up}}\) is \(\{1, 2, \ldots, k - 1, k\}\) because we do not try to delete \(k\). In addition, \(S_{\text{right}} = \{r_{2k}\}\) and \(L_{\text{labeled}}\) has one node labeled by \(\{r_{k+1}, r_{2k+1}\}\). The others are
After several steps, we reach the deNM-tree $T_2$ shown in Figure 29. Up to now, we have tried application of the $\otimes$-elimination $3k$ times. In the active node, we have

$$\mathcal{S}_{\text{right}} = \{r_{k+1}, \ldots, r_{2k}\}$$
$$\mathcal{L}_{\text{up}} = 2k$$
$$\mathcal{S}_{\text{up}} = \{1, 2, \ldots, k-1, k\}.$$ 

The others are empty. The for the only element $2k$ in $\mathcal{L}_{\text{up}}$ application of the $\otimes$-elimination rule is tried and succeeds. After several steps, we finally obtain one node tree. In total from the start to the end, we have tried application of the $\otimes$-elimination $4k$ times, which is linear.
Example 3.2 As seen previously, the deNM-tree \( T(\Theta_3) \) shown in Figure 23 is obtained from MLL proof structure \( \Theta_3 \) shown in Figure 22, which is not an MLL proof net. When we choose the node \( n \) labeled by \( \{r_2\} \) as the starting rule, since \( L_{\text{down}} = 1 \) for \( n \), i.e., it is not empty, the local jump rule is applied to \( \&\text{-link} \ 1 \). After two applications of the union rule, in the current active node, \( L_{\text{down}}, L_{\text{labeled}}, L_{\text{right}}, \) and \( L_{\text{up}} \) become all empty. So the output is no.

4 Concluding Remarks

In this paper we have accomplished a new linear-time correctness condition of unit-free MLL proof nets based on the rewriting system over trees. Among known linear-time correctness conditions of MLL, ours is definitely simplest. There are some future research directions.

- Extensions of our result to variants like noncommutative fragments or extensions like MALL or MELL of MLL.
- Implementation issues: In particular, to some extent it may be possible to have several active nodes in a DR-tree and to exploit parallelism using a multi-core processor.
- Application to proof search: In \( [8] \) in order to search MLL proof nets for a given MLL formula, backtracking mechanism and a naïve implementation of de Naurois and Mogbil’s correctness condition are combined. Our result may be used to obtain more elegant implementations for MLL proof search. That was our original motivation for this work.
- Mechanical formalization using your favorite interactive theorem prover.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to algorithms, third edition, 3rd ed., The MIT Press, 2009.
[2] V. Danos, La logique linéaire appliquée à l’étude de divers processus de normalisation (principalement du lambda-calcul), Ph.D. thesis, Université Paris 7, 1990.

[3] V. Danos and R. Regnier, The structure of multiplicatives, Archive for Mathematical Logic 28 (1989), 181–203.

[4] P. J. de Naurois and V. Mogbil, Correctness of linear logic proof structures is NL-complete, Theoretical Computer Science 412 (2011), 1941–1957.

[5] Harold N. Gabow and Robert Endre Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (1985), no. 2, 209–221.

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

[7] Stefano Guerrini, Correctness of multiplicative proof nets is linear, Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (Washington, DC, USA), LICS ’99, IEEE Computer Society, 1999, pp. 454–.

[8] Satoshi Matsuoka, Proof Net Calculator, https://staff.aist.go.jp/s-matsuoka/PNCalculator/index.html, 2017.

[9] Andrzej S. Murawski and C.-H. Luke Ong, Dominator trees and fast verification of proof nets, Proceedings of the Fifteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2000), IEEE Computer Society Press, June 2000, pp. 181–191.