EFFICIENT PROOF NET VERIFICATION AND SEQUENTIALIZATION

ARTHUR VALE

Abstract. When Girard introduced Linear Logic [Gir87], he also developed a novel syntax for writing proofs, proof nets, which enjoys many desirable properties for representing proofs. Some important technical issues in handling proof nets is that of correctness (whether a syntactically sound proof structure is logically sound) and sequentialization (constructing a sequent proving the same conclusion as that of the proof net). This article surveys the literature on efficient proof net correctness verification and sequentialization for the multiplicative fragment of Linear Logic without units. First, we briefly present the original correctness criterion by Girard, which yields exponential time correctness verification in the size of the proof structure. Then, we show that the most famous criterion for correctness, due to Danos and Regnier in [DR89], is equivalent to that of [Gir87], yielding exponential time correctness verification over the number of par nodes in the proof structure. Next, we show that a development on ideas from [DR89] yields correctness verification and sequentialization in quadratic time over the number of links in the proof structure. Finally, we introduce the main idea of Guerrini from [Gue11] which leads to linear time complexity over the size of the proof structure for simultaneous correctness verification and sequentialization.

Contents

Date: August 14th, 2018.

1. Introduction

1.1. **Prerequisites.** The author expects that most, if not all, of the main thread of the article should be within reach of a mathematically mature and interested reader. Cursory knowledge about sequent calculus is assumed, and familiarity from Graph Theory will prove useful. Remarks and digressions often target readers familiar with particular subjects, but those may be glossed over without harm for the overall understanding of the article.

1.2. **Linear Logic.** Linear Logic was properly introduced in 1987 by Girard in [Gir87]. Previous work already hinted at the full system but lacked a satisfying theory. Linear Logic has the following connectives: *linear implication* $\multimap$, *linear negation* $^\perp$, the conjunctions *times* $\otimes$ and *with* $\&$, the disjunctions *plus* $\oplus$ and *par* $\parr$, the exponentials *of course* $!$ and *why not* $?$, and the units for times $\mathbf{1}$, with $\top$, plus $\mathbf{0}$ and par $\bot$. In the following we give some background knowledge about Linear Logic, and some of its applications.

1.2.1. *Semantics.* In the 1987 article, Girard motivates Linear Logic through a couple of different interpretations. The first, in Tarskian style, interprets formulas as particular subsets of a commutative monoid with an identified set of phases. He calls this a phase semantics. The second, in Heyting style, follows the tradition of interpreting proofs as functions over a Scott domain. However, this interpretation makes the innovation of using coherent spaces (binary qualitative domains) introduced in 1986 in [Gir86]. These are simpler than Scott domains in that finitary notions are actually finite in a specific sense. Proofs in Linear Logic are then interpreted categorically over the category of coherent spaces (as morphisms between coherent spaces), akin to how lambda calculus is often embedded in closed Cartesian categories. Surprisingly, this the interpretation works remarkably well in the realm of coherent space; in particular, linear implication is interpreted as linear stable functions over coherent spaces.

Both interpretations yield the same sequent calculus, in the sense that the sequent calculus for Linear Logic defined by Girard in [Gir87] is sound and complete with respect to both interpretations. Having that, we need not focus on how Linear Logic is interpreted but rather focus on the logical system itself, whose sequent calculus is introduced in the following.

1.2.2. *Its Position Within Logic.* Often, Linear Logic is praised for simultaneously being an extension of intuitionistic and classical logic, by which we mean that faithful translations from the latter to the former are possible by defining classical implication within Linear Logic with the use of the exponentials, and then restricting the logic to specific fragments [Gir87]. These translations are remarkable in that, unlike the translations between intuitionistic and classical logic, the disjunctive property holds within the corresponding fragment of Linear Logic, and linear negation remains involutive [Gir11]. Linear Logic is, in fact, much more expressive than the other two systems, and one can even translate Girard's system **F** (second-order propositional natural deduction) within Linear Logic [Gir87] (with quantitatives). This translation is semantically sound with respect to proof nets for second-order predicate linear calculus, with the caveat of using an inelegant syntactical feature, boxes. Linear Logic really succeeds in decomposing important logical features in its key fragments (multiplicative, additive and exponential) and

it is perhaps from this that stems the success in the translation of *old* logic. Furthermore, Linear Logic enjoys the Haupstatz without much trouble, and hence it isn't just another modal logic (since often modal logics do not).

1.2.3. *Applications.* Linear Logic also seems to achieve an untouched ground of constructiveness. In particular, Linear Logic can be used as a logic of resources, states, time and most important, computation. Indeed, there are several important applications to computer science, specifically to concurrent systems and functional languages. Most approachable, a series of papers by Philip Wadler, written in a conversational tone while still presenting the full technical complexities of the system, is great at advertising these applications [Wad90][Wad91][Wad92][Wad93a][Wad93b]. Linear Logic is also of great interest to functional programming languages because it is able to keep track of states in a satisfying manner, yielding functional programming language designs that don't have to trick the underlying logical system in order to change the machine state. This is elegantly exemplified in [Wad90] by a class of vectors that support in-place updates with no need for garbage collection or reference counting. This is due mainly to the property that in Linear Logic, a formula, unless modified by an exponential, may only be used exactly once, due to the lack of weakening and contraction in the fragment without exponentials. This stands in heavy contrast to the *old* logics.

1.3. **Linear Sequent Calculus.** For reference, below we give a list of the rules of the sequent calculus for Linear Logic. They are presented as a right calculus[1] for simplicity, as in [Gir11]. The full system is presented for the interested reader, but only the multiplicative fragment will be used in what follows. As is typical in proof-theory literature, capital Greek letters are used for lists of formulas while capital Latin characters are used for formulas (which are formed by logical constants, negation and atomic formulas denoted by lowercase Latin characters). We use σ for a permutation, while $A[t/x]$ is the result of substituting the variable x for the term t in the formula A. Finally, $\Gamma^\perp$ has already been defined, and $!\Gamma$ and $?\Gamma$ are defined analogously. As usual, *identity* only need hold for atomic formulas, and we impose the usual restriction that x is not free in any of the formulas in Γ in the *for all* rule. Linear negation is an operator over atomic formulas and therefore is

[1] i.e. one uses symmetricity of negation in Linear Logic, as in classical logic, to turn all bilateral sequents $\Gamma \vdash \Delta$ into $\vdash \Gamma^\perp, \Delta$, where $\Gamma^\perp$ denotes the list of the negations of the formulas in Γ. This significantly diminishes the number of rules in the calculus since there are no longer any elimination rules, which are quite essentially left rules.

defined modulo De Morgan's Law for other formulas. That is:

$$\mathbf{1}^\perp := \bot \qquad \bot^\perp := \mathbf{1}$$

$$\mathbf{0}^\perp := \top \qquad \top^\perp := \mathbf{0}$$

$$(p)^\perp := \bot \qquad (p^\perp)^\perp := p$$

$$(A \otimes B)^\perp := A^\perp \parr B^\perp \qquad (A \parr B)^\perp := A^\perp \otimes B^\perp$$

$$(A \oplus B)^\perp := A^\perp \,\&\, B^\perp \qquad (A \,\&\, B)^\perp := A^\perp \oplus B^\perp$$

$$(!A)^\perp := ?A^\perp \qquad (?A)^\perp := !A^\perp$$

$$(\exists x A)^\perp := \forall x A^\perp \qquad (\forall x A)^\perp := \exists x A^\perp$$

Linear implication is, in fact, redundant and can be encoded by

$$A \multimap B := A^\perp \parr B$$

This is justified by either showing the equivalence under an interpretation of Linear Logic, or as can be seen in the following bilateral derivation from the sequent calculus with $\multimap$:

$$\dfrac{\dfrac{\dfrac{A \multimap B \vdash}{A \vdash B}\,(\multimap\mathrm{E})}{\vdash A^\perp, B}\,(\bot\mathrm{I})}{\vdash A^\perp \parr B}\,(\parr\mathrm{I})$$

Since the derivation is invertible, this justifies the definition. Finally, it is important to note that there is no introduction rule for $\mathbf{0}$. The motivation for the names multiplicative, additive and exponential come from their interpretations either in phase semantics or in coherent spaces. In the case of the multiplicative and additive fragments, one can see that the premises for multiplicative rules are multiplicative (the contexts are independent), while they are additive for the additive rules (the contexts are dependent). Full Linear Logic, when in sequent style, is typically called **LL**, while the multiplicative fragment (the identity rules, with the structural rules and the multiplicative rules) is called **MLL**, and the multiplicative with additives fragment (all the rules in **MLL** as well as the additive rules) is called **MALL**. Finally, the unit-free multiplicative system is denoted $\mathbf{MLL}^-$, and the unit-free multiplicative with additives fragment is denoted $\mathbf{MALL}^-$. The full sequent calculus follows below.

1.3.1. *Identity.*

$$\frac{}{\vdash A^\perp, A}\,(\mathit{identity}) \qquad \frac{\vdash \Gamma, A \qquad \vdash A^\perp, \Delta}{\vdash, \Gamma, \Delta}\,(\mathit{cut})$$

1.3.2. *Structure.*

$$\frac{\Gamma}{\vdash \sigma(\Gamma)}\,(\mathit{exchange})$$

1.3.3. *Multiplicatives.*

$$\frac{}{\vdash \mathbf{1}}\,(one) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\,(false)$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, A \otimes B, \Delta}\,(times) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,(par)$$

1.3.4. *Additives.*

$$\frac{}{\vdash \Gamma, \top}\,(true) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,(with)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,(left\ plus) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,(right\ plus)$$

1.3.5. *Exponentials.*

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,(of\ course) \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\,(dereliction)$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A}\,(weakening) \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\,(contraction)$$

1.3.6. *Quantifiers.*

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x A}\,(for\ all) \qquad \frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x A}\,(there\ exists)$$

It is convenient to always think of a formula as rather representing an occurrence of the formula, and the remaining of the paper will take this convention.
Finally, we present a derivation of transitivity of linear implication in sequent calculus style, in order to have an example to recourse to when necessary. We omit applications of *exchange* in all sequent calculus derivations.

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

Notice though that this is far from the only possible derivation of transitivity of linear implication. Trivially, one may apply superfluous exchanges, but the problem is much deeper, as seen in yet another derivation:

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

It is easy to see that there are many other commutations possible. This problem, at least within $\mathbf{MLL}^-$ (and in a more recent development even in $\mathbf{MALL}^-$), is solved by using a different syntax for representing proofs.

1.4. **Proof Nets.** Proof nets, also introduced in [Gir87], provide a novel syntax for $\mathbf{MLL}^-$. In truth, proof structures are the terms under this syntax, and the correct proof structures are called proof nets. In [Gir87], Girard shows that proof nets (under the correctness criterion there defined) for $\mathbf{MLL}^-$, often called $\mathbf{PN_0}$, are both sound and complete with respect to $\mathbf{MLL}^-$. Namely, every sequent derivation corresponds to some proof net that has the same conclusions as that of the derivation, and every proof net corresponds to some sequent calculus derivation with the same conclusions as that of the proof net. In fact, Girard also shows that the representation is unique in the sense that there is a single proof net for each set of conclusions provable in $\mathbf{MLL}^-$. Furthermore, if one allows for cuts in proof nets, normalization can be done linearly over the size of the proof net. Moreover, it enjoys both confluence and Church-Rosser, and therefore proof-nets are strongly normalizable. Due to uniqueness, proof nets provide a sensible medium for proving the Haupstatz, since its proofs tend to be mostly about presenting procedures to handle commutations, which simply don't exist in proof nets.

In [DR89], a correctness criterion simpler than that of [Gir87] was shown to be equivalent to that introduced in the latter. The Danos-Regnier criterion is in fact so simple that it seems like most of the logical content is irrelevant for checking correctness. Indeed, in [Gue11] (introduced beforehand by Guerrini in a paper from 1999 and expanded in 2011), this is indeed shown to hold some truth. Guerrini introduces a criterion based on contracta of proof nets which stems from Danos' PhD thesis from 1990 [Dan90]. He then shows it is equivalent to a parsing algorithm, with the result that correctness in $\mathbf{PN_0}$ is merely a syntactical property. Furthermore, Guerrini presents the first linear time algorithm for $\mathbf{PN_0}$ correctness and sequentialization, thus showing that proof nets yield no loss of information in comparison to the sequential representation.

Extending the notion of a proof net for the full logical system seems difficult to do[2]. In fact, finding a simple correctness criterion for extensions of $\mathbf{MLL}^-$ is a persistent problem in the theory. Some notable attempts are seen in: [Gir91], where proof nets are extended to quantifiers; and in [Hug05], where they are extended to all of **MLL**; and in [HVG03], where an extension to $\mathbf{MALL}^-$ is presented, which is invariant upon commutations, unlike an earlier proposal by Girard in [Gir96]. Regarding complexity, there is hardly any hope that correctness can be done efficiently even for **MLL**, since it has been shown that the problem is in fact **NP**-hard [Laf93].

Before we give a formal definition of a proof structure, we first give an informal run-through. Proof nets are assemblies of three kinds of links: axiom links, times links and par links (four if we include cut links). They have the following shapes (Fig. 1). Each with the implied connection to the logical system from the labels. This said, here is the proof net for transitivity of linear implication (Fig. 2).

In the theory of proof nets, as hinted by the previous discussion, there are three main desirable properties. A correctness criterion for proof structures so that the class of correct proof structures is in fact exactly the class of those proof structures whose conclusions are provable in the logical system one intends to represent. That is, the proof nets must enjoy logical soundness and completeness. Furthermore, it is important to develop a procedure to both convert sequents to proof nets and proof nets to sequents. Finally, unless proof nets are unique for each set of provable

[2]without resorting to sequential segments inside of proof nets, such as boxes.

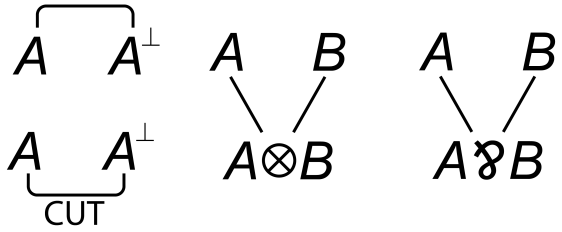

FIGURE 1. From left to right: identity link; times link; par link.

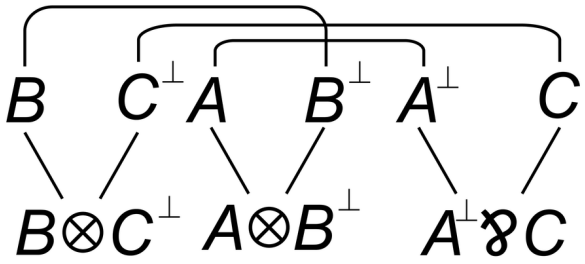

FIGURE 2. Proof of transitivity

conclusions, they are no better in terms of commutations than sequents. Therefore this sort of invariance is fundamental for their usefulness.

We finish this introduction with a formal definition of a proof structure.

Definition 1.1 (link)**.** A *link* is a pair $\beta \triangleright \alpha$ of sets of vertices, β being the set of *premises* and α the set of *conclusions* of the link, such that α and β are disjoint ($\alpha \cap \beta = \emptyset$) and not mutually empty ($\alpha \cup \beta \neq \emptyset$).

Definition 1.2 (proof structure)**.** A *proof structure* $G = (V, L)$ is a hypergraph whose finite set of vertices V is identified with occurrences of formulas in Linear Logic or occurrences of an identified cut symbol, together with a set L of links (the edges of the hypergraph), where each link has the form

$$\triangleright A, A^{\perp}, \text{ or } A, B \triangleright A \parr B, \text{ or } A, B \triangleright A \otimes B, \text{ or } A, A^{\perp} \triangleright \text{CUT}, \text{ or CUT} \triangleright$$

where A and B are formula occurrences from V. Furthermore, a proof structure must satisfy the conditions
(i) every vertex in V is the conclusion of exactly one link in L.
(ii) the premise of at most one link in L.
(iii) there exists at least one vertex in V that is not a premise of any link in L,
(iv) every cut symbol is the premise of a dummy link (CUT $\triangleright$).
Any formula occurrence in V that is not a premise of any link in L is called a *conclusion* of G. If α is the set of all conclusions of G, then one writes $G \vdash \alpha$.

Times and cut links are called binary links, and par links are called unary links due to their role in the logical system. A closely related notion to that of proof structures is that of syntactically valid substructures.

Definition 1.3 (link structure)**.** A pair $G' = (V', L')$ of vertices $V' \neq \emptyset$ and links $L' \neq \emptyset$ over V' is a *link structure* if there exists a proof structure $G = (V, L)$ such that $V' \subset V$ and $L' \subset L$. A formula that is not a conclusion of any link in L' is called a *premise* of G', and if β is the set of all premises of G', and α is the set of all conclusions of G' then one writes $G : \beta \vdash \alpha$.

If $G' \subset G$, where G is a proof structure, then G' is called a *link substructure* of G, and if G' is a proof structure as well then G' is also called a *proof substructure*.

For now, we leave it to the reader to consolidate the formal definition of a proof structure with the geometrical representation provided above. In particular, even though a link of te form $\alpha \triangleright \beta$ is an edge $\alpha \cup \beta$ in the hypergraph, premises and conclusions of each edge are retrievable from the extra restrictions on a proof structure. Furthermore, we remark that considering cut as a formal symbol in the system is merely a technicality that proves very useful. In the graph induced by a proof structure, a cut and a times link have the same topology, with the added technicality of a dummy link at every cut so that no cut is a conclusion of a proof structure. We will see soon that, in fact, cuts and times are no different in what regards correctness. Link structures can be represented by simple graphs instead of hypergraphs (by connecting each premise of its links to the conclusion of the appropriate link, see *infra*) so that we are fluid in which representation to use. For completion, the following is a formal definition of a proof net in (simple) graph representation.

Definition 1.4. The *graph induced by a proof structure* $G = (V, L)$ is a simple graph with the same vertices as G and with the following edges (we denote the existence of an edge connecting v and u by $v \frown u$ and the absence of such an edge by $v \smile u$):
Axiom Link: If $\triangleright A, A^\perp \in L$ then $A \frown A^\perp$.
Binary Links: If $A, B \triangleright C$ is a binary link in L (either a cut or a times link), then $A \frown C$ and $B \frown C$.
Unary Links: If $A, B \triangleright A \parr B$ is an unary link in L then, $A \frown A \parr B$ and $B \frown A \parr B$.
For any other pair of vertices u and v, $u \smile v$.

Of course, if the binary link is a times, then C is the times of A and B, while if it is a cut link then B must be an occurence of $A^\perp$ and C is the cut symbol. We note that except for axiom links (which lack a premise and therefore have their conclusions connected), and dummy links (which do not yield any edges), all that this construction does is connect premises to conclusions.

These definitions are only enough to define a syntactically correct structure. In order to guarantee logical correctness one needs to have some further restriction on which proof structures will actually represent proofs. This is what we pursue next.

2. Trips on Proof Nets

2.1. **Trips.** The correctness criterion introduced by Girard in [Gir87] relies on the concept of a trip over a proof structure. Informally, a trip can be interpreted as a particle travelling along a path or a package of information travelling through some channel or network. One first chooses a starting vertex and direction for the trip and then, up to a choice of a sort of orientation of travel for each times and par link, the trip follows a deterministic route, eventually coming back to the initial point by traveling in the same direction. The number of formulas that the particle

passed through is the length of the trip. Correctness is then verified by checking that any trip has length twice the size of the proof structure. We now give a formal definition of the concept of a trip over a proof structure, and other relevant notions. We refrain from giving a proof in full formality that being correct in terms of trips is indeed equivalent to being logically correct, which would involve a definition of time, as well as showing that trips are indeed cyclic, concepts that have mostly technical value despite the relevance of the results.

Definition 2.1 (switching)**.** A *(full) switching* of a proof structure G is a choice of labeling L (standing for a left switch) or R (a right switch) for every par, times and cut link.

In graph representation, a switching induces a specific graph, in the sense that from such a graph one is able to retrieve the switching and the original graph.

Definition 2.2. The *graph induced by a switching* of a proof structure $G = (V, L)$ is a graph with the same vertices as G and the following edges:

Axiom Link: If $\vdash A, A^{\perp} \in L$ then $A \frown A^{\perp}$.
Binary Links: If $A, B \vdash C$ is a binary link in L (either a cut or a times link), then $A \frown C$ and $B \frown C$.
Unary Links: If $A, B \vdash A \parr B$ is an unary link in L then, if the link is switched to L, $A \frown A \parr B$ and $B \smile A \parr B$. If the link is switched to R, $A \smile A \parr B$ and $B \frown A \parr B$.
For any other pair of vertices u and v, $u \smile v$.

Note that the graph induced by a switching is the same as that in the graph representation of a proof structure, except for the removal of one of the edges in par links. Only par switches are considered, which is reminiscent of the discussion in the end of Section 2. Since switchings of binary links do not change the induced graph we may restrict our attention to only the par switches. We commit to that by restricting switches to par links in the future. These graphs will become quite important in future sections, but the original definition of trip did not rely on them.

Definition 2.3 (trip)**.** A vertex in a proof structure $G = (V, L)$ may be traversed by a particle in one of two directions. Such a traversal is called a *motion*. If $A \in V$ is a vertex being transversed "upward", then we denote the motion through A by $A^{\wedge}$. Else, if the vertex is being transversed "downward" then we denote the motion by $A^{\vee}$.

Given a switching of G, we denote a segment of a trip by a list of comma-separated motions, where time increases from left to right; i.e. If a motion happens at time t the next motion in the list happens at time $t+1$. In the case of an upward motion, the next motion traverses a premise of which the formula just traversed is a conclusion, and in the case of a downward motion, the next motion traverses the conclusion to which the formula just traversed is a premise. Lastly, the trip does not repeat any motions, that is, it represents one lap through the proof structure. In the case of an upward motion, where the premise is not unique, the switching decides which premise is next.

The definition of a *trip* through each link follows, where each trip is defined by the following sequence of motions over each link, and ... elide the appropriate omitted segment of the proof over the proof structure, which must mutually agree

with the other trip definitions as well as with the restrictions above. Each link has two possible trips, depending on the point of entrance.

Axiom link ($\triangleright A, A^{\perp}$):

$$\ldots, A^{\wedge}, A^{\perp\vee}, \ldots \text{ or } \ldots, A^{\perp\wedge}, A^{\vee}, \ldots$$

Times Link ($B \triangleright A \otimes B$):
If the switch is set to L then,

$$\ldots, A^{\vee}, A \otimes B^{\vee}, \ldots, A \otimes B^{\wedge}, B^{\wedge}, \ldots \text{ or } \ldots, B^{\vee}, A^{\wedge}, \ldots$$

If the switch is set to R then,

$$\ldots, A^{\vee}, B^{\wedge}, \ldots \text{ or } \ldots, B^{\vee}, A \otimes B^{\vee}, \ldots, A \otimes B^{\wedge}, A^{\wedge}$$

Cut Link ($A, A^{\perp} \triangleright \text{CUT}$ and $\text{CUT} \triangleright$): Notice that since every cut symbol is the premise of a dummy link, we may treat cut links and dummy links together.
If the switch is set to L then,

$$\ldots, A^{\vee}, \text{CUT}^{\vee}, \text{CUT}^{\wedge}, A^{\perp\wedge}, \ldots \text{ or } \ldots, A^{\perp\vee}, A^{\wedge}, \ldots$$

If the switch is set to R then,

$$\ldots, A^{\vee}, A^{\perp\wedge}, \ldots \text{ or } \ldots, A^{\perp\vee}, \text{CUT}^{\vee}, \text{CUT}^{\wedge}, A^{\wedge}, \ldots$$

Par Link ($A, B \triangleright A \parr B$):
If the switch is set to L then,

$$\ldots, A^{\vee}, A \parr B^{\vee}, \ldots, A \parr B^{\wedge}, A^{\wedge}, \ldots \text{ or } \ldots, B^{\vee}, B^{\wedge}, \ldots$$

If the switch is set to R then,

$$\ldots, A^{\vee}, A^{\wedge}, \ldots \text{ or } \ldots, B^{\vee}, A \parr B^{\vee}, \ldots, A \parr B^{\wedge}, B^{\wedge}, \ldots$$

For an example, consider the proof net for transitivity of linear implication. If we consider the switching where each times and cut switch is set to L and each par switch is set to R then a trip starting with the motion $A^{\vee}$ yields the following trip:

$$A^{\vee}, A{\otimes}B^{\perp\vee}, A{\otimes}B^{\perp\wedge}, B^{\perp\wedge}, B^{\vee}, B{\otimes}C^{\perp\vee}, B{\otimes}C^{\perp\wedge}, C^{\perp\wedge}, C^{\vee}, A^{\perp}{\parr}C^{\vee}, A^{\perp}{\parr}C^{\wedge}, C^{\wedge},$$
$$C^{\perp\vee}, B^{\wedge}, B^{\perp\vee}, A^{\wedge}, A^{\perp\vee}, A^{\perp\wedge}$$

Notice that by the remark that all trips are in cyclic, it follows that the number of motions in a trip is bounded above by twice the number of vertices in the proof structure. Furthermore, the remark that starting point or orientation yield the same trip up to a cyclic permutation implies that the number of motions in a trip is invariant under starting point and direction. This justifies why the following concepts are well-defined.

Definition 2.4 (short/long trip)**.** The *length* $|t|$ of a trip t is the number of motions in the trip. If n is the number of formulas and CUT occurrences in a proof structure G then a trip is said to be *short* if $|t| < 2n$ and *long* if $|t| = 2n$.

2.2. **Trip Correctness.** Finally, we can define Girard's original correctness criterion.

Definition 2.5 (trip correctness)**.** A proof structure is said to be trip-correct if for every switching of the proof structure, the corresponding trip is a long trip.

We then see that the proof structure for the transitivity of linear implication has 9 formulas and the trip presented before has length 18, so it is indeed a long trip. One would need to check the remaining seven switchings in order to prove trip-correctness, but, in fact, since we already presented sequent calculus derivations for the statement, the following two results from [Gir87] show that the proof structure given above is a proof net.

Theorem 2.6 (Trip Correctness)**.** *If π is a proof of $\vdash \Gamma$ in $\boldsymbol{MLL}^-$, then there is a trip-correct proof structure $G(\pi) \vdash \Gamma$.*

Theorem 2.7 (Sequentialization)**.** *If $G \vdash \Gamma$ is a trip-correct proof structure, then there exists a proof π of $\vdash \Gamma$ in $\boldsymbol{MLL}^-$ such that $G(\pi) = G$.*

This establishes a criterion that is both necessary and sufficient for logical correctness and therefore, we may indeed define proof nets according to trip-correctness. Namely:

Definition 2.8 (proof net)**.** A proof net is a trip-correct proof structure.

We do not give proofs of either the Trip Correctness theorem or the Sequentialization theorem. The proof of the Trip Correctness theorem is rather simple. It proceeds by induction over the bottom-most formula of the sequent calculus derivation in the natural way. Namely, one introduces the link corresponding to the last rule of the sequent, and applies induction in order to obtain the premise proof nets. The base case, that of an axiom rule, is trivial. Then, one needs to prove that the inductively constructed proof structure only admits long trips, which is lengthy but straight forward.

Sequentialization turns out to be much more subtle. The argument proceeds by induction as well, and uses the idea of splitting proof nets. These splittings get complicated when all conclusions of a proof net are the conclusion of a times (cut) link. In this case, the choice of which conclusion to use for the splitting is crucial, for there are proof nets such that a particular choice of times splitting does not split the proof net into two disconnected proof subnets, which is necessary for the induction argument. In order to avoid this, Girard used a fairly subtle and long argument to prove that, if a proof net has no conclusion that is a result of a par link, then there is always a choice of a times (cut) link such that splitting at that times (cut) link yields two disjoint proof subnets. This is called the splitting theorem. With the splitting theorem, an inductive procedure akin to reversing that used for the trip correctness theorem works smoothly. One then proves the result by showing that the constructed sequent does convert into the original proof net with the inductive procedure from the Trip Correctness Theorem, thus establishing Sequentialization. Notice that it is from these non-deterministic choices of where to split next in the inductive procedure for Sequentialization that several sequents that conclude the same set of formulas can be obtained from a single, unique, proof net.

With respect to complexity, trip correctness is not a sensible correctness criterion for proof structures. Indeed, in order to show that a proof structure G is trip-correct, one needs to check the length of a trip for all possible switchings, and there are $2^{n_1+n_2}$ of those, where n_1 is the number of unary switches and n_2 the number of binary switches in a switching of the proof structure. Since performing the trips is linear in the size of the proof structure, this yields an $O(|G|2^{n_1+n_2})$ time bound. It is not hard to see that, in fact, only $2^{n_1+n_2-1}$ switchings need to be checked, since flipping the switch for every times and cut link gives a trip that has the same sequence of motions as that before switching but in reverse time, so each check of a switching actually also checks the corresponding flipped switching. This does not yield any improvements in terms of asymptotic behavior though. The criterion is also not particularly satisfying, since it relies on rather *ad hoc* definitions of these trips over the proof structure. Furthermore, the fact that flipping times (cut) switches does not yield new information about correctness also hints at some redundancy in the criterion. Finally, the way the switching is defined for a par link, where one essentially ignores one of the premises, also suggests that it is the difference between trips where different sets of par premises are ignored that really matter for correctness. This is elegantly captured by the correctness criterion defined by Danos and Regnier in [DR89], to be presented next.

3. Proof Nets as Trees

The correctness criterion introduced by Danos and Regnier is quite simple. It relies on the simple graph representation of proof structures, which was introduced in Section 1 and Section 2.

Definition 3.1 (tree correctness)**.** A proof structure G is *tree-correct* if every graph induced by a switching of G is a tree.

Below follows the definition of the restricted notion of a switching to par links.

Definition 3.2 (switching)**.** A *(par) switching* of a proof structure G is a choice of labeling L (standing for a left switch) or R (a right switch) for every par link.

In particular, checking par switchings of a proof structure is enough to establish tree correctness. Hence, in the side of complexity, if n_1 is the number of par links in a proof structure G, then one only needs to check whether 2^{n_1} graphs are trees. Since checking if a graph is a tree is $O(n)$ checking for tree correctness directly is $O(|G|2^{n_1})$, an improvement over trip correctness.

To finish this section, the equivalence of tree correctness and trip correctness is established.

Theorem 3.3 (Tree Correctness)**.** *A proof structure is tree-correct if and only if it is trip-correct.*

Proof. First, we take G, a proof-structure, in graph representation. Given a switching s one may modify the graph induced by the switching, $T(G, s)$, by making it a directed graph $T'(G, s)$ and then substituting two directed edges for each undirected edge. In case of an axiom link or a unary link we give them reverse orientations, and in case of $A \frown C$ and $B \frown C$ both due to the same binary link, $T'(G)$ is such that $A \to C$, $C \to B$ and $B \to A$ in the case of an L switch on the binary link and $A \to B$, $B \to C$ and $C \to A$ in the case of an R switch. Motions are defined

as expected in order to agree with the definition given in Section 2. It is a simple task to show that trips on proof structures agree with proofs on these graphs, and furthermore, that the expected correspondence between motions and directed edges holds. We note that any long trip visits each edge of $T'(G)$ exactly once due to this correspondence. Hence, G is trip-correct if and only if every trip over $T'(G)$ is a long trip.

With that established, we first prove the forward direction by induction. The base case of just an axiom link is easily established by noticing there is only the empty switching, and one possible trip $A^\wedge, A^{\perp\vee}, A^{\perp\wedge}, A^\vee$ which is long (it is clearly tree-correct). Fix a switching s of the proof structure $G = (V, L)$ and then let $T(G, s)$ be the graph induced by the switching. Since by assumption $T(G, s)$ is a tree, it has at least one leaf v. Since v has degree 1, v must be a conclusion. If v is a conclusion of a par link, retract the graph by joining this vertex with its parent u (and removing the edge between them) to obtain a retract R. Then apply induction to obtain a long trip $\ldots, u^\vee, u^\wedge, \ldots$ over the corresponding R' as defined above, which must have length $2(|G| - 1)$. Restore v and add the edges $u \to v$ and $v \to u$ for a graph T'. It is clear that $T' = T'(G, s)$. Then, $\ldots, u^\vee, v^\vee, v^\wedge, u^\wedge, \ldots$ is easily seen to be a valid trip and its length is $2(|G| - 1) + 2 = 2|G|$ so that it is, in fact, long. One need not consider the case where v is the premise of a par link, since every par link also has a leaf that is a conclusion. If v was the conclusion of a binary link (including the case where v is the cut symbol), remove v as well as any edges added due to the binary link forming a graph R. Since by assumption, $T(G, s)$ is a tree and v had degree 2, this removal splits $T(G, s)$ into two connected components in R. Apply induction to each connected component to obtain a trip $\ldots_1, l^\vee, l^\wedge, \ldots_2$ for the connected component R_l where the left premise l of the link lies, and $\ldots_3, r^\vee, r^\wedge, \ldots_4$ for the connected component R_r of the right premise, both long trips over the connected components of R'. Add v together with the directed edges induced by the switch of the binary link to obtain a graph T'. Since we added exactly the edges that are missing from $T'(G, s)$, it is clear that $T' = T'(G, s)$. Now, if the link has an L switch, then $\ldots_1, l^\vee, v^\vee, v^\wedge, r^\wedge, \ldots_4, \ldots_3, r^\vee, l^\wedge, \ldots_2$ is a valid trip over $T'(G, s)$, and if the switch is R $\ldots_1, l^\vee, r^\wedge, \ldots_4, \ldots_3, r^\vee, v^\vee, v^\wedge, l^\wedge, \ldots_2$ is a valid trip over $T'(G, s)$. In both cases, the constructed trips have length

$$2(|R_l|) + 2(|R_r|) + 2 = 2(|R_l| + |R_r|) + 2 = 2(|G| - 1) + 2 = 2|G|.$$

Hence both trips are long ones. If all leaves of $T(G, s)$ are conclusions of axiom links, then G can only involve axiom links. In this case, either the only link in G is an axiom link (handled in the base case), or there are several axiom links. In the latter case, $T(G, s)$ is disconnected, and therefore not tree-correct.

Now, for the backward direction, it is enough to notice that any long trip over $T'(G, s)$ visits every vertex exactly twice, and in particular, that $T'(G, s)$ is strongly connected. But $T'(G, s)$ is weakly connected if and only if $T(G, s)$ is connected by construction. Now, to each conclusion of $T(G, s)$ add a dummy pendant vertex and then let $L(G, s)$ be the adjoint of this graph. In the adjoint, vertices are links and edges are formulas (with the technical exception of the vertices corresponding to the added pendant edges at conclusions). Define $L'(G)$ by doubling every edge and giving them reverse orientations. Motions over $L'(G)$ are again defined so as to agree with those in Section 2, and it is again easy to show that trips over $L'(G, s)$ are equivalent to trips over G. Now, given a long trip over $L'(G, s)$ form L' by

gluing edges representing the same formula that are traversed by the trip. It is clear then that $L' = L(G, s)$ and therefore L' is acyclic. □

This proven, we have that indeed, we may define proof nets instead as tree-correct proof structures. Despite that, checking for correctness is still exponentially hard. In the following section, we greatly improve the complexity by giving yet two other correctness criteria that allow for quadratic correctness checking as well as sequentialization.

4. Contractability and Parsing

Contracting structures, contracting subnets that are already known to be tree-correct, is the fundamental technique behind a quadratic correctness check, as well as sequentialization. The idea is originally from Danos but is greatly developed by Guerrini in [Gue11].

4.1. **Contractibility.** One starts by defining links to substitute substructures.

Definition 4.1 (star link)**.** Given a set of premises β and a set of conclusions α such that $\alpha \cap \beta = \emptyset$ and $\alpha \cup \beta \neq \emptyset$, a *star link* is a link $\beta \triangleright^* \alpha$. A *contracted proof (link) structure* is a proof (link) structure that along with all the restrictions of a proof (link) structure, also admits star links. A *par (full) switching* of a proof structure is a choice of labels L and R for each unary (and binary) link. The *graph induced by a contracted structure* is constructed the same way and satisfies the same constraints, as that for a proof structure with the additional constraint that if $\beta \triangleright^* \alpha \in G$ is a link in the contracted structure, then there exists a permutation $u_0, u_1, \ldots, u_k$ of α, β such that $u_i \frown u_{i+1}$ for every $i \in \{0, 1, \ldots, k-1\}$.

Tree correctness for a contracted proof structure is still the same. That is, a contracted proof structure is tree-correct when the graph induced by any switching is a tree. The fundamental result for contracted proof structures that allows one to decompose correctness to substructures is the following. In the statement, $L_1 L_2$ is the concatenation of the list of links L_1 with the list of links L_2, and $V|_L$ is the restriction of the set of vertices V to only those that occur in the links in the list L.

Lemma 4.2. *Let $G = (V, L_0 L_1)$ be a contracted link structure such that $G_0 = (V|_{L_0}, L_0)$ is a tree-correct link structure with $G_0 : \beta \vdash \alpha$. Then, G is tree-correct if and only if $G' = (V, L_1(\beta \triangleright^* \alpha))$ is tree-correct.*

Proof. Fix a switching s of G and consider $T(G, s)$. $T(G, s)$ must have $T(G_0, s)$ as a subgraph by definition of G, and since G_0 is tree-correct, $T(G_0, s)$ is a tree as a subgraph of $T(G, s)$. Now, take the link structure $S = (\beta \cup \alpha, \beta \triangleright^* \alpha)$ and fix a switching for it. Substitute $T(S, s)$ for $T(G_0, s)$ in $T(G, s)$ forming T'. By definition of the edges induced by a star link, $T(S)$ is a tree. Hence, the substitution substitutes a tree subgraph of a tree for a path between the same vertices, and therefore the T' must still be a tree. But $T' = T(G')$ for some switching. It is evident that the reverse substitution with the assumption that G' is tree-correct also yields the graph induced by some switching over G. Therefore, there exists a switching that induces a non-tree graph over G if and only if there exists a switching that induces a non-tree graph over G'. □

Lemma 4.2 motivates the definition of the following contraction rules over lists of links. They are defined so that contraction rules introduce star links that correspond to link substructures.

Definition 4.3 (contraction rules)**.** Let L be a list of links.
(0-ary links)

$$L(\triangleright A, A^\perp) \to_c L(\triangleright^* A, A^\perp) \text{ or } L(\text{CUT } \triangleright) \to_c L(\text{CUT } \triangleright^*)$$

(unary links) if the occurrence of $A \parr B$ in the right-side of the unary rule is not in β:

$$L(\beta \triangleright^* A, B, \alpha)(A, B \triangleright A \parr B) \to_c L(\beta \triangleright^* A \parr B, \alpha)$$

(binary links)

$$L(A, B \triangleright A \otimes B) \to_c L(A, B \triangleright^* A \otimes B) \text{ or } L(A, A^\perp \triangleright \text{CUT}) \to_c L(A, A^\perp \triangleright^* \text{CUT})$$

(star links) if $\alpha_1 \cap \beta_2 = \alpha_2 \cap \beta_1 = \emptyset$ and $\alpha_1 \cup \alpha_2 \cup \beta_1 \cup \beta_2 \neq \emptyset$

$$L(\beta_1 \triangleright^* A, \alpha_1)(A, \beta_2 \triangleright^* \alpha_2) \to_c L(\beta_1, \beta_2 \triangleright^* \alpha_1, \alpha_2)$$

Let $G = (V, L)$ and $G' = (V, L')$ be such that $L \to_c L'$, then we write $G \to_c G'$. Furthermore, if $G = (V, L_0 L_1)$ and $G_0 = (V|_{L_0}, L_0)$ and $G_1 = (V|_{L_1}, L_1)$ we take the liberty of lifting concatenation from sets of links to link structures by $G = G_0 G_1$. These contraction rules do only introduce star links when there are corresponding link substructures and preserve correctness, as we will show through the following sequence of lemmas.

Lemma 4.4. *Let G be a contracting link structure such that $G \to_c^* G'$ (that is, several contraction steps lead from G to G'). Then, for all star links $\beta \triangleright^* \alpha$ in G', there exists a unique tree-correct link substructure $G_0 = (V|_{L_0}, L_0) : \beta \vdash \alpha$ of G such that $L_0 \to_c^* \beta \triangleright^* \alpha$, and if $G = G_0 G_1$ and $G' = (V, (\beta \triangleright^* \alpha) L_1)$, then $G_1 \to_c^* G_1'$.*

Proof. To prove that G_0 exists, it is enough to proceed by induction over the length of the $G \to_c^* G'$ reduction. The base case, that of a length 0 reduction, is easily seen to hold by taking an empty link substructure as G_0. For the inductive step, it is evident that one may take the last contraction rule used and that defines how to build G_0 from the inductively obtained link substructure G_0'. For instance, in the case of the 0-ary, unary and binary links one adds the link just removed to G_0', and in the case of a star contraction one merely joins the removed conclusion/premise. That the G_0 so constructed is tree-correct also follows from induction by showing that no contraction rule adds a cycle, as long as there were not already any cycles. For uniqueness, assume there is another tree-correct $G_0' = (V|_{L_0'}, L_0')$ satisfying the statement of the lemma. Then, one has that $L_0' = (L_0 \cap L_0')(L_0' \setminus L_0)$, so let $L_0'' = L_0 \cap L_0'$. Now, since any vertex is a conclusion/premise of at most one link, and since L_0 is enough for $G_0 : \beta \vdash \alpha$, then is must be that $G_0'' = (V|_{L_0''}, L_0'')$ is such that $G_0'' : \beta \vdash \alpha$. But then, $L_0' \setminus L_0 = \emptyset$ (otherwise, any switching would have the vertices involved in L_0'' and those involved in $L_0' \setminus L_0$ in different connected components, contradicting tree correctness). Hence, $L_0' \subset L_0$. Repeating the argument with the roles of L_0 and L_0' interchanged, one obtains $L_0 = L_0'$. Now, since both G_0 and G_0' admit $\beta \vdash \alpha$, their vertex sets must be the same and hence, $G_0 = G_0'$. □

Lemma 4.5. *Let G be a contracting link structure such that $G \to_c^* G'$. Then, G is tree-correct if and only if G' is tree-correct.*

Proof. Just apply Lemma 4.2 and Lemma 4.4 to each star link in G'. □

Hence, contraction rules preserve tree correctness. Furthermore, the following lemma shows that in fact, tree-correct structures are strongly contractable.

Lemma 4.6. *Let $G : \beta \vdash \alpha$ be a tree-correct contracting link structure. Then, G has a unique normal form $\beta \triangleright^* \alpha$, and G contracts to $\beta \triangleright^* \alpha$ in finitely many steps.*

Proof. Any sequence of contraction rule applications is finite, for any of the first three contraction rules in Definition 4.3 reduce the number of axiom links, binary links or unary links by one and add at most a single star link, while the star link rule reduces the number of star links by one.
Suppose first that $\beta = \emptyset$ and let $G \to_c^* G'$, where G' is a normal form. G' has no axiom links or binary links, as otherwise they would be reducible in a single step and G' would not be a normal form. Furthermore, by Lemma 4.4, G' is tree-correct and contains a link of the form $\triangleright^*\gamma$ (by taking the maximal element of the links ordered by being "immediately below", which is a partial order due to a tree-correct structure being acyclic). Now, $\gamma = u_1, \ldots, u_k, \alpha'$, where $\alpha' \subset \alpha$ and none of the u_i for $i \in \{1, \ldots, k\}$ are conclusions of G. But, since G' is in normal form, any u_i can only be a premise of an unary link $u_i, v_i \triangleright u_i \parr v_i$ such that $v_i \notin \gamma$, otherwise it would be immediately reducible by the unary contraction rule. Hence, any switching that has the link $u_i, v_i \triangleright u_i \parr v_i$ labeled as R is such that $v_i \frown u_i \parr v_i$. On the other hand the switch of $\triangleright^*\gamma$ is in another connected component of the graph induced by such a switching. But since G' must be tree-correct, it follows that $k = 0$. Now, if $\beta = u_1, \ldots, u_k \neq \emptyset$, just add an axiom link $\triangleright u_i, u_i^\perp$ for each $i \in \{1, \ldots, k\}$ forming G_β. By what was just proven, the only normal form of G_β is $\triangleright^*\alpha(u_1^\perp, \ldots, u_k^\perp)$. Now, if $G = (V, L)$ and given a contraction $G \to_c^* G'$ to normal form of G, one obtains one for G_β by;

$$L_\beta = L(\triangleright u_1, u_1^\perp, \ldots, \triangleright u_k, u_k^\perp) \to_c^* L'(\triangleright u_1, u_1^\perp, \ldots, \triangleright u_k, u_k^\perp) \to_c^* \triangleright^*\alpha(u_1^\perp, \ldots, u_k^\perp).$$

Now, since the occurrences in the conclusion of α come from the added axiom links, by Lemma 4.2 it follows that $G' \to_c^* \beta \triangleright^* \alpha$. □

Now if, in fact, one considers a proof structure G, instead of a link structure, then, by Lemma 4.6 it must be the case that if G is tree-correct, then the only normal form of G under contraction rules is of the form $\triangleright^*\alpha$ where α is the set of conclusions of G. Hence one can define a new correctness criterion.

Definition 4.7. A proof structure $G = (V, L)$ such that $G \vdash \alpha$ is contractible when $L \to_c^* \triangleright^*\alpha$.

Now we may finally establish contractibility as a correctness criterion.

Theorem 4.8. *A proof structure is tree-correct if and only if is contractible.*

Proof. The forward direction comes from Lemma 4.6. The backward direction from Lemma 4.4 and the trivial remark that $G' = (V, \triangleright^*\alpha)$ is tree-correct. □

We will see in the following that any contraction $G \to_c^* \triangleright^*\alpha$ yields a sequentialization of $G \vdash \alpha$.

4.2. **Parsing.** The proof of Theorem 4.8 suggests a particular contraction strategy: only reduce when a link with no premises is available. This is possible in a proof structure because, as has been previously remarked, ordering formula occurrences by the "immediately below" strategy yields a partial order and therefore one may always reduce by finding maximal elements in this partial order. That is:

Definition 4.9 (parsing rules)**.** Let L be a list of links.
(0-ary links)

$$L(\triangleright A, A^\perp) \to_p L(\triangleright^* A, A^\perp)$$

(unary links) if the occurrence of $A \parr B$ in the right-side of the unary rule is not in β:

$$L(\triangleright^* A, B, \alpha)(A, B \triangleright A \parr B) \to_p L(\triangleright^* A \parr B, \alpha)$$

(binary links)

$$L(\triangleright^* \alpha_A, A)(\triangleright^* \alpha_B, B)(A, B \triangleright A \otimes B) \to_p L(\triangleright^* \alpha_A, \alpha_B, A \otimes B)$$

$$\text{or } L(\triangleright^* \alpha_A, A)(\triangleright^* \alpha_{A^\perp}, A^\perp)(A, A^\perp \triangleright \text{CUT}) \to_p L(\triangleright^* \alpha_A, \alpha_{A^\perp}, \text{CUT})$$

(star links) If $\alpha \neq \emptyset$,

$$L(\triangleright^* \alpha, \text{CUT})(\text{CUT} \triangleright) \to_p L(\triangleright^* \alpha)$$

Definition 4.10 (parsed proof structure)**.** A parsed proof structure is a contracted proof structure with only premiseless star links.

By checking that indeed, parsing rules are just contraction rules that always make use of premiseless star links, one establishes the following lemma.

Lemma 4.11. *If $G \to_p^* G'$, then $G \to_c^* G'$*

Proof. By inspection, as explained in the above remarks. □

It then follows from the proof of Lemma 4.6 that parsing is terminating for any parsed proof structure. Moreover, by Lemma 4.5, that tree-correctness is invariant under parsing.

Lemma 4.12. *There is no infinite parsing of a proof structure.*

Proof. Just the beginning of Lemma 4.6, and noticing that one never uses the assumption that the proof structure is tree-correct to prove that contraction terminates. □

Lemma 4.13. *Let G be a parsed proof structure such that $G \to_p^* G'$. Then G is tree-correct if and only if G' also is.*

Proof. Just an instantiation of Lemma 4.5, which is justified by Lemma 4.11. □

With these two results the only remaining result needed to establish well-foundedness of parsing as a correctness criterion is the proof that indeed, the only normal form of a proof net $G \vdash \alpha$ under the parsing rules is $\triangleright^* \alpha$. First, we need an auxiliary result.

Lemma 4.14. *Let $G = (V, L)$ be a tree-correct parsed proof structure. If n_* is the number of star links in G, n_0 the number of axiom links, and n_2 the number of binary links, then $n_* + n_0 - n_2 = 1$.*

Proof. From graph theory, we know that an acyclic graph with n vertices and e edges has $n - e$ connected components. Now, if c_* is the number of conclusions of star links, then for the graph induced by a switching of G, each axiom link contributes one vertex; each binary or unary link contributes another vertex; and each star link contributes as many conclusions as it has. Hence, if n_1 is the number of unary links:

$$n = 2n_0 + n_1 + n_2 + c_*$$

Furthermore, each axiom link contributes a single edge, as does each unary link. A binary link on the other hand contributes two edges. A star link on k conclusions (because parsing only ever creates premiseless star links) contributes $k - 1$ edges, for a path between vertices as defined in the graph induced by a switching of a contracted proof structure. Hence

$$\begin{aligned} e &= n_0 + n_1 + 2n_2 + \sum_{\triangleright^*\alpha, |\alpha| = k} (k - 1) \\ &= n_0 + n_1 + 2n_2 + \left(\sum_{\triangleright^*\alpha, |\alpha| = k} k \right) - \left(\sum_{\triangleright^*\alpha \in L, |\alpha| = k} 1 \right) \\ &= n_0 + n_1 + 2n_2 + c_* - n_* \end{aligned}$$

Therefore:

$$n - e = (2n_0 + n_1 + n_2 + c_*) - (n_0 + n_1 + 2n_2 + c_* - n_*) = n_* + n_0 - n_2$$

but since any tree-correct proof structure is connected, we get that

$$n_* + n_0 - n_2 = 1.$$

□

Lemma 4.15. *Let $G \vdash \alpha$ be a tree-correct parsed proof structure. Then G has a unique normal form $\triangleright^*\alpha$ under parsing.*

Proof. Let $G = (V, L)$. Suppose $G \to_p^* G'$ with G' a normal form under parsing rules. From Lemma 4.5 and Lemma 4.11, it follows that G' is tree-correct. Then let $G' = (V, L')$ and decompose L' into $L' = L_0(\triangleright^*\alpha_1)(\triangleright^*\alpha_2)\dots(\triangleright^*\alpha_l)$, where L_0 has no star links. Since G' is normal, similarly to in the proof of Lemma 4.6, one has that L_0 has no axiom links (immediately reducible). Furthermore, there is no occurrence of a cut symbol in $\alpha_1 \cup \ldots \cup \alpha_l$, since then it would be immediately reducible by the star link parsing rule, and for any α_i and for any $u \in \alpha_i$ such that $u, v \triangleright u \parr v \in L_0$, one has that $v \notin \alpha_i$. Now, assume $G_0 \neq \emptyset$. Let $\alpha_i^1 = \{u \in \alpha_i : \exists v \in V \text{ s.t. } u, v \triangleright u \parr v \in L_0\}$ and similarly let $\alpha_i^2 = \{u \in \alpha_i : u \text{ is a premise of a binary link in } L_0\}$. By the restrictions just given, $\alpha_i = \alpha_i^1 \cup \alpha_i^2$. Since $G_0 \neq \emptyset$, $\alpha_i^2 \neq \emptyset$ for any i, since otherwise, by the same argument from Lemma 4.6, there would be a unary link such that setting a particular label for the switch disconnects the graph induced by the switching. Furthermore, if $G_0 \neq \emptyset$, there are at least l conclusions of star links that are premises of binary links. Then, by Lemma 4.14, $l = n_2 + 1$, so that, by the pigeonhole principle, at least one binary link has both its premises, say u and v, being conclusions of star links, say $u \in \alpha_i$

and $v \in \alpha_j$. If $i = j$, the graph induced by the proof structure can't have been tree-correct. If $i \neq j$, the star links to which u and v are conclusions together with the binary link for which they are premises are immediately reducible, contradicting that G' is a normal form. Hence $G = \emptyset$, and therefore it must be that $l = 1$ and $\alpha_1 = \alpha$. □

So indeed, one can use parsing as a correctness criterion.

Definition 4.16. A proof structure $G = (V, L)$ such that $G \vdash \alpha$ is parsable if $G \to_p^* \triangleright^* \alpha$.

Theorem 4.17. *A proof structure is tree-correct if and only if it is parsable.*

Proof. The forward direction is a consequence of Lemma 4.13 and the backward direction comes from Lemma 4.15. □

Contraction and parsing give us two particular strategies for checking for correctness, namely, by reducing a proof structure to normal form and then checking if it has the necessary shape. We can quickly see that both approaches are quadratic in complexity. This is because all one needs to do is search for a redex, which is $O(n)$, and then reduce it. Since every rule reduces the number of links in the proof structure that can be used for a future redex by at least 1, it follows that only $O(n)$ reductions need to be done for a quadratic time correctness-checking algorithm. Parsing can, in fact, be done in $O(|G| \log |G|)$ by exploiting the fact that all redexes involve maximal elements for searching for the next redex. But parsing is in fact much stronger, for it provides a sequentialization algorithm as well.

4.3. **Sequentialization.** The following theorem, a classic result in the theory of proof nets, can be proved by making use of Theorem 4.17. Indeed, the whole argument hinges on the status of the reduction rules for parsing as actual parsing rules.

Theorem 4.18 (Sequentialization)**.** *A proof structure G is a proof net if and only if there is a sequent calculus derivation for G in **MLL**$^-$.*

Proof. From Theorem 4.17, we may assume that G is parsable. The proof proceeds by induction on the size of the proof structure G. Suppose first that G is a proof net. If G has a single link, it must be an axiom link $\triangleright A, A^\perp$, and therefore the identity axiom can be instantiated as a proof of $\vdash A, A^\perp$. Now, let $G \to_p^* \triangleright^* \alpha$ be a reduction of G proving it is parsable, and assume that whenever the binary parsing rule is used to eliminate a cut, the next rule is the rule that eliminates the corresponding CUT $\triangleright$ link. We lose no generality in making this assumption since that is the only rule that can remove such a link. We consider the last rule of the reduction in cases:
(star links) In this case, by the assumption on the removal of cut links, the before last reduction step must be:

$$(\triangleright^* \alpha_A, A)(\triangleright^* \alpha_{A^\perp}, A^\perp)(A, A^\perp \triangleright \mathrm{CUT})(\mathrm{CUT}\ \triangleright) \to_p (\triangleright^* \alpha_A, \alpha_{A^\perp}, \mathrm{CUT})(\mathrm{CUT}\ \triangleright)$$

where $\alpha_A \alpha_{A^\perp} = \alpha$. This is then followed by the removal of the CUT $\triangleright$ link. Now, by Lemma 4.4 together with Lemma 4.11, it follows that there exists a unique proof structure $G_A \vdash \alpha_A, A$, as well as $G_{A^\perp} \vdash \alpha_{A^\perp}, A^\perp$, such that $G_A \to_p^* \triangleright^* \alpha_A, A$ and $G_{A^\perp} \to_p^* \triangleright^* \alpha_{A^\perp}, A^\perp$. In particular they are both parsable. Then, by induction, there exists sequents π_A and $\pi_A^\perp$ proving $\vdash \alpha_A, A$ and $\vdash \alpha_{A^\perp}, A^\perp$ respectively. By

using π_A and $\pi_{A^\perp}$ as premises for a cut inference rule, one obtains a proof of $\alpha_A\alpha_{A^\perp} = \alpha$.
(binary links) It can't be the case that the last rule is a binary link rule parsing the link $A, A^\perp \triangleright \mathrm{CUT}$ by the assumption that those are followed by a star link parsing rule. Therefore, the last rule in this case must be:

$$(\triangleright^*\alpha_A, A)(\triangleright^*\alpha_B, B)(A, B \triangleright A \otimes B) \to_p L(\triangleright^*\alpha_A, \alpha_B, A \otimes B),$$

and it must be the case that $\alpha_A\alpha_B(A \otimes B) = \alpha$. Now, as in the case for a star link, let π_A and π_B be sequents proving $\vdash \alpha_A, A$ and $\vdash \alpha_B, B$, respectively. One constructs a sequent for $\alpha_A\alpha_B(A \otimes B) = \alpha$ by using π_A and π_B as left and right premises (respectively) of a times inference rule.
(unary links) In this case, the last step of parsing is:

$$(\triangleright^* A, B, \alpha')(A, B \triangleright A \parr B) \to_p \triangleright^* A \parr B, \alpha'.$$

It must be that $(A \parr B)\alpha' = \alpha$. By the same argument used in the previous cases, one finds a sequent π' proving $\vdash A, B, \alpha'$ and then, using π' (up to an exchange rule) as the premise of a par inference rule, one obtains a sequent proving $\vdash A \parr B, \alpha'$.
(0-ary link) This coul only be the last parsing rule in the reduction if the proof net was such that $G \vdash A, A^\perp$, but this is the base case.
It follows by induction that any proof net is sequentializable.

Now, assume π is a sequent calculus derivation of $\vdash \alpha$. A sequent of length 1 can only be an identity axiom, which is handled by a proof net with a single axiom link. Now, for the inductive step, let π' be the sequent obtained by removing the last inference rule of π and assume that π' proves $\vdash \alpha'$. Apply induction to obtain a proof net $G' \vdash \alpha'$. By the inductive hypothesis, $G' \to_p^* \triangleright^*\alpha'$. Now, add to G' a par link $A, B \triangleright A \parr B$ if the last rule of π was a par rule inferring $A \parr B$. Add a times link $A, B \triangleright A \otimes B$ if it was a times rule inferring $A \otimes B$ and add a cut link $A, A^\perp \triangleright \mathrm{CUT}$ (together with a dummy link CUT $\triangleright$) if it was a cut rule that cuts A with $A^\perp$. It is readily seen that adding those links will still satisfy all the restrictions of Definition 1.2, because G is well-founded as a proof structure and G' has as conclusions the premises of the last inference rule in π. Finally, to show that G is parsable, one applies the same parsing rules as the reduction $G' \to_p^* \triangleright^*\alpha'$ in the same order to obtain a parsing of $G \to_p^* (\triangleright^*\alpha')l$, where l is the added link(s) as described above. Now, if l is a par link, one extends the reduction just constructed by a unary link parsing rule; if l is a times link, one extends the reduction by a binary link parsing rule removing the times link; if l is a cut link, one extends the reduction by a binary link parsing rule removing a cut, followed by a star link parsing rule removing the dummy link. It is quickly seen, by the previous remarks on G' and α', that these parsing rules will be possible in each case. Therefore, G is parsable and thus G is a proof net. □

By the proof of Theorem 4.18, one can simultaneously check for correctness, by using the parsing criterion, and construct a corresponding sequent for the proof net, by adding the relevant inference rule to the sequent each time a parsing step is performed. Hence, by our remarks at the end of section 4.2, it follows that sequentialization can also be performed in quadratic time.

5. Toward Linear Time

Although being able to check for correctness and perform sequentialization in quadratic time is quite good, one can in fact perform them in linear time as discussed in the introduction. The linear time algorithm is a more involved version of the algorithm presented in this section, with additional technical considerations over its complexity. The key idea is, instead of incrementally parsing the proof structure, just mark those vertices that would have been parsed already. This is done by a unification-style algorithm. Rephrasing the problem in this way shows that, in fact, one may use disjoint union-find to efficiently perform this sort of unification. Due to space constraints, we only present this reformulation from parsing to unification, which gives a flavor of the way toward true linear time. It is by considering more closely the data structure used to store the graph and a more careful argument about the particular instance of union-find for this algorithm, that one is able to perform both correctness and sequentialization in linear time.

The idea is to start from the top of the proof structure (axiom links) and go down, labeling the vertices as part of the proof substructure that contains that axiom link. When there are conflicts, one must make sure to resolve them without breaking the invariant that the currently disjoint sets of labels do represent proof subnets. The key for this intuition is that if the unification rules for such conflicts mimic the parsing rules, being unifiable should be equivalent to being parsable.

Definition 5.1 (partial unifier)**.** A *partial unifier* for a proof structure $G = (V, L)$ is a pair (μ, π) of a marking function $\mu : V \to [-1, n_0 - 1]$, where the interval in the codomain is taken over the integers, n_0 is the number of axiom links in L, and π is a partition of $[0, n_0 - 1]$ represented as $\pi = (t_1; t_2; \ldots; t_l)$ with each t_i a set of integers in the range $[0, n_0 - 1]$.

In addition, partial unifiers support the following operations. In the range of μ, -1 represents an unlabeled node. If $\mu(V) = [0, h)$, then $\text{next}(\mu) = h$. $\mu[\alpha \mapsto i]$ is the marking function μ' defined by $\mu'(u) = i$ if $u \in \alpha$ and $\mu'(u) = \mu(u)$ otherwise. $\pi(i)$ is the least element of the set containing i in the partition π. Let $i \in t \in \pi$ and $j \in t' \in \pi$ and $\pi = (t; t'; t_1; \ldots; t_l)$; then $\pi[i = j]$ is the partition $\pi' = (t \cup t'; t_1; \ldots; t_l)$.

Given a partial unifier $U = (\mu, \pi)$ and set t in π, $G[\mu, t]$ is the link substructure of G formed by all links whose premises and conclusion are labeled by an integer in t.

The key properties one wants the unification system to enjoy are that, at any point of unification, $G[\mu, t]$ is a proof substructure, and that $G[\mu, t] \vdash \alpha$ should be equivalent to the existence of a parsing $G[\mu, t] \to_p^* \triangleright^* \alpha$. This is achieved by making the unification rules closely mimic the effect of the parsing rules.

Definition 5.2 (unification rules)**.** Fix a proof structure $G = (V, L)$. Then the unification rules for the proof structure G currently admitting a partial unifier (μ, π) are defined by:

(0-ary) If $\triangleright A, A^\perp \in L$ and $\mu(u_1) = \mu(u_2) = -1$ then

$$(\mu, \pi) \to_u (\mu[\{A, A^\perp\} \mapsto \text{next}(\mu)], (\pi, \text{next}(\mu)))$$

(unary link) if $A, B \triangleright A \parr B \in G$ and $\pi(\mu(A)) = \pi(\mu(B)) = i$ then

$$(\mu, \pi) \to_u (\mu[A \parr B \mapsto i], \pi)$$

(binary link) if $A, B \triangleright C \in G$ and $j = \pi(\mu(A)) < \pi(\mu(B)) = i$ then

$$(\mu, \pi) \to_u (\mu[C \mapsto j], \pi[i = j])$$

One can easily see that the unification rules just given are, in a sense, equivalent to the parsing rules from Definition 4.9. Namely, a unification step can only be taken when a parsing rule would create a star link under the same conditions, thinking of $G[\mu, t]$ (for $t \in \pi$) as the star links in a equivalent contracted proof structure. Furthermore, no rule is necessary for a dummy link, since there is only one sensible choice of label for a dummy link: the label received by its only premise. Now, we define the equivalent notion of correctness for unification.

Definition 5.3 (unifiable proof structure)**.** A proof structure G with n_0 axiom links is *unifiable* if, by starting with a partial unifier (μ, π), where μ is the identically -1 marking function and π is the partition of the empty set, there is a unification such that $(\mu, \pi) \to_u^* (\mu', \pi')$, where μ' is a total function onto $[0, n_0 - 1]$ and $\pi' = (0, \dots, n_0 - 1)$.

The natural next step would be to prove that, indeed, being unifiable can be used as a correctness criterion.

Theorem 5.4. *A proof structure G is unifiable if and only if it is a proof net.*

The proof of theorem 5.4 is not particularly interesting, since unification is constructed so that it works, as is evident from the previous remarks. The proof is therefore omitted. The interested reader will find it in [Gue11]. It merely uses the fact that parsability is well-founded as a correctness criterion. For one direction, one proceeds by induction, substitutes every $G[\mu, t_i] \vdash \alpha_i$ for $t_i \in \pi$ by $\triangleright^* \alpha_i$ and applies the induction hypothesis. On the other direction one shows, by the clear equivalence between parsing steps and unification steps, that if $G \to_p^* G'$ with $G' = G_0(\triangleright^* \alpha_1) \dots (\triangleright^* \alpha_l)$ where G_0 has no star links, then there exists a partial unifier (μ, π), $\pi = (t_1, \dots, t_l)$, such that $G[\mu, t_i] = G_i$.

It is not hard to see how unification rules can be turned into an algorithm. One merely keeps seeking the next place to apply a unification step until no such steps are possible any more. For the reading computer scientist, it is clear that, in fact, a partial unifier (μ, π) is just an instance of a disjoint set union-find data structure (π) and a mapping (μ). Disjoint set union-find is a well-studied data structure, and if one does use it to implement π then any m successive applications of the partition operations take time $O(\alpha(m, n)m)$ where α is the inverse of the Ackerman function and n is the number of times one added a new label to the partition. In particular, n is the number of axiom links in the proof structure, and m is the number of times a unifier rule is applied, which by similar considerations to those given for parsing, is in fact linear over the size of G. Hence, by using union-find, one obtains a pseudo-linear cost for performing a unification. Despite that, when either a unary link has distinct labels in its premises or a binary link has the same tokens in its premises, one needs to re-route unification until some unification step unifies the two premises of the unary link. In the case of the binary link, one must try unification somewhere else in the proof net. If this process of re-routing unification is not done efficiently, by having a suitable data structure to schedule the next re-routes, then it will take linear time to search for the next place to re-route and the algorithm will therefore be pseudo-quadratic. These are the additional considerations that need to be taken

to improve to a pseudo-linear algorithm for unification. Further considerations can lead to a true linear time algorithm, as seen in [Gue11].

Acknowledgments

I thank Sarah Reitzes for all her support through the Summer leading to this survey article, feedback and willingness to learn. It is a pleasure to thank Prof. Kurtz for suggesting me the topic of Linear Logic. Finally, my gratitude to Prof. May and his unwavering commitment to encourage the development new mathematicians.

References

[Dan90] V Danos. Une application de la logique linéaire à l'Ètude des precessus de normalisation (principalement du λ-calcul). *PhD Thesis*, 1990.

[DR89] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[Gir86] Jean-Yves Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir91] Jean-Yves Girard. Quantifiers in linear logic II. *Atti del Congresso Nuovi problemi della logica e della filosofia della scienza*, II:79–89, 1991.

[Gir96] Jean-Yves Girard. Proof nets: the parallel syntax for proof theory. *Logic and Algebra*, 1996.

[Gir11] Jean-Yves Girard. *The Blind Spot*. European Mathematical Society, Zürich, 2011.

[Gue11] Stefano Guerrini. A linear algorithm for mll proof net correctness and sequentialization. *Theoretical Computer Science*, 412:1958–1978, 2011.

[Hug05] Dominic Hughes. Simple multiplicative proof nets with units. *arXiv:math/0507003*, 2005.

[HVG03] Dominic Hughes and Rob Van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *Proceedings 18th Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, 2003.

[Laf93] Y. Lafont. From proof nets to interaction nets. *Advances in Linear Logic (Proceedings of the Workshop on Linear Logic Ithaca, New York, June 1993)*, 1993.

[Wad90] Philip Wadler. Linear types can change the world! *Programming Concepts and Methods*, 1990.

[Wad91] Philip Wadler. Is there a use for linear logic? *Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, 1991.

[Wad92] Philip Wadler. There's no substitute for linear logic. *8'th International Workshop on the Mathematical Foundations of Programming Semantics*, 1992.

[Wad93a] Philip Wadler. A syntax for linear logic. *Ninth International Conference on the Mathematical Foundations of Programming Semantics*, 1993.

[Wad93b] Philip Wadler. A taste of linear logic. *Mathematical Foundations of Computing Science*, 1993.