Proof Nets and Boolean Circuits*

Kazushige Terui
National Institute of Informatics, Japan
terui@nii.ac.jp

Abstract

We study the relationship between proof nets for mutiplicative linear logic (with unbounded fan-in logical connectives) and Boolean circuits. We give simulations of each other in the style of the proofs-as-programs correspondence; proof nets correspond to Boolean circuits and cut-elimination corresponds to evaluation. The depth of a proof net is defined to be the maximum logical depth of cut formulas in it, and it is shown that every unbounded fan-in Boolean circuit of depth n, possibly with $st\mathrm{CONN}_2$ *gates, is polynomially simulated by a proof net of depth* $O(n)$ *and vice versa. Here,* $st\mathrm{CONN}_2$ *stands for st-connectivity gates for undirected graphs of degree 2. Let* APN^i *be the class of languages for which there is a polynomial size,* $\log^i$*-depth family of proof nets. We then have* $APN^i = AC^i(st\mathrm{CONN}_2)$.

1. Introduction

Proof nets [4, 2, 6] are a parallel syntax for logical proofs, which has arisen in the study of linear logic. Traditional proofs involve lots of inessential sequential information in proof construction that makes cut-elimination a global and sequential procedure. The proof net syntax remedies traditional ones by removing those innessential sequentialities. An outcome is a local and parallel cut-elimination procedure. So far it has been most successful for the *multiplicative fragment of linear logic* (**MLL**), that is classical propositional logic without weakening nor contraction, though it is being refined and extended in various directions [5, 7, 11].

Boolean circuits (see [17, 1, 16] for instance) are one of the standard models of parallel computation, which has been studied mainly in computational complexity theory. The model has been extensively used in analyzing parallel complexity of functions. Most notably, it has provided a number of nontrivial lowerbounds on the complexity of practically interesting functions, such as parity [3], majority [14], modulo-p [15] and st-connectivity [8].

The aim of this paper is to establish a formal connection between these two models of parallel computation. More precisely, we wish to relate them in the style of the proofs-as-programs correspondence, where proofs are identified with circuits and cut-elimination is identified with evaluation of circuits. Actually, a first step has already been taken by [12, 13], where it is shown that every Boolean circuit of size s can be simulated by an **MLL** proof (coded by a linear lambda term) of size $O(s)$. Since cut-elimination in **MLL** can obviously be done in quadratic time, the simulation has resulted in the P-completeness of cut-elimination in **MLL**.

Two important issues are left untouched, however. First, the notion of *depth* is not taken into account at all. Without depth, the simulation is not very interesting, because the point of parallel computation is to achieve a speed-up by use of multiple processors, and in the case of Boolean circuits it is nothing but depth that reflects the speed of computation. Second, the converse simulation from proofs to circuits is not given at all. Although cut-elimination could be simulated by a Turing machine and a Turing machine could in turn be simulated by a circuit, that would not give efficiency in terms of parallel computation. In particular, the circuit thus obtained would be as deep as the time required for sequential cut-elimination. In other words, the main advantage of parallel computation would be entirely lost.

This paper attempts to offer a better link between proofs and circuits. The first step of improvement is to use proof nets, which are intrinsically parallel, instead of lambda terms, which are primarily sequential. We then introduce a suitable notion of depth for proof nets and give simulations in both directions which preserve depth-efficiency.

After giving some background on Boolean circuits (Section 2), we present an unbounded fan-in variant of **MLL**, called **MLLu**, and consider the proof nets for it (Section 3). The reason for considering **MLLu** is that proof nets seem to better relate to circuits in the unbounded fan-in setting. The depth of a proof net is defined to be the maximum logical depth of cut formulas in it.

*Partially supported by Grant-in-Aid for Scientific Research, MEXT, Japan.

Several encodings of Boolean functions are given next (Section 4). Of particular interest are the encodings of the parity function PARITY and the majority function MAJ. To show the correctness of PARITY, we employ some graph theoretic argument which seems to be authentic to proof nets. To encode MAJ, we exploit higher order facility of proof nets; in some sense, proof nets are capable of representing a sort of "higher order gate" whose output is not a Boolean value but a Boolean function. The majority function is smoothly coded by the medium of a higher order functional. We also give an encoding of $st\text{CONN}_2$ (st-connectivity gates for undirected graphs of degree 2) along the same line, and prove that *every unbounbded fan-in circuit of size s and depth d, possibly with $st\text{CONN}_2$ gates, can be simulated by a proof net of size $O(s^5)$ and depth $O(d)$.*

To give the converse simulation, we present a parallel cut-elimination procedure according to which the number of reduction steps is bounded in terms of depth (rather than size) and furthermore, each reduction step is simulated by a constant depth circuit (Section 5). To deal with axiom cuts, we consider a global reduction rule of *tightning*, and implement it by means of $st\text{CONN}_2$ gates. We then prove that *every proof net of size s and depth d (which represents a Boolean function) can be simulated by an unbounded fan-in circuit with $st\text{CONN}_2$ gates of size $O(s^4)$ and depth $O(d)$.*

A hierarchy *APN* of proof net complexity classes is defined, fully analogously to the *AC* hierarchy (Section 6). For $i \geq 0$, let APN^i be the class of those languages for which there is a polynomial size, $\log^i$-depth family of proof nets. From what precedes, it follows that $APN^i = AC^i(st\text{CONN}_2)$, and therefore the union *APN* of the hierarchy amounts to the class *NC*, which is reasonably considered to be the class of effectively parallelizable functions, for which a parallel computer could achieve a dramatic speed-up over standard sequential computers.

2. Boolean Circuits

In this section, we briefly recall the definition of Boolean circuits and several known facts on them (see [17, 1, 16] for further information).

Definition 1 (Boolean circuits) *A basis $\mathscr{B}$ is a set of Boolean functions. A* Boolean circuit *C with n inputs (and one output) over $\mathscr{B}$ is a labeled, directed acyclic graph. The nodes of in-degree 0 are called* input nodes, *and are labeled with $x_1, \ldots, x_n, 0, 1$. Non-input nodes are called* gates *and are labeled with a Boolean function from $\mathscr{B}$ whose arity coincides with its in-degree (also called its fan-in). There is a unique node of out-degree 0, that is called the* output node.

The size *is the number of gates, and the* depth *is the length of the longest path from an input to the output node.*

A circuit C with n inputs *accepts* a word $w = i_1 \cdots i_n \in \{0,1\}^n$ if C evaluates to 1 when $i_1, \ldots, i_n$ are assigned to $x_1, \ldots, x_n$. C *accepts* a language $X \subseteq \{0,1\}^n$ if C accepts w just in case $w \in X$ for every $w \in \{0,1\}^n$. A single circuit only works on inputs of fixed length. If one wants to solve a problem for inputs of arbitrary length, one needs to have an infinite *family* of circuits, one for each input length. We say that a family $\{C_n\}_{n\in\mathbb{N}}$ of circuits *accepts* a language $X \subseteq \{0,1\}^*$ if C_n accepts $X \cap \{0,1\}^n$ for every $n \in \mathbb{N}$.

Besides standard Boolean connectives, we consider the following functions:

- *Parity*: $\text{PARITY}^n(x_1, \ldots, x_n) = 1$ iff the sum of $x_1, \ldots, x_n$ is odd.
- *Majority*: $\text{MAJ}^n(x_1, \cdots, x_n) = 1$ iff the sum of $x_1, \ldots, x_n$ is greater than or equals to $n/2$.
- *st-Connectivity*: Given $E = \{x_{i,j}\}_{1\leq i<j\leq n}$ coding an undirected graph G over vertices $\{1, \ldots, n\}$, $st\text{CONN}^n(E) = 1$ iff there is a path from vertex 1 to n in G. (Without loss of generality, we may assume that G does not contain a loop. Hence we do not take the values $x_{i,i}$ into account.)
- *st-Connectivity for graphs of degree d*: $st\text{CONN}^n_d$ is analogous to $st\text{CONN}^n$, but works only for undirected graphs of degree at most d. (When E codes a graph of degree greater than d, $st\text{CONN}^n_d(E) = 0$.)

In this paper, we are concerned with the *standard unbounded fan-in basis* $\mathscr{B}_1 = \{\neg\} \cup \{\bigwedge_n, \bigvee_n\}_{n\in\mathbb{N}}$, possibly equipped with a family $\mathscr{F}$ of Boolean functions such as $\text{MAJ} = \{\text{MAJ}^n\}_{n\in\mathbb{N}}$ and $st\text{CONN}_d = \{st\text{CONN}^n_d\}_{n\in\mathbb{N}}$. A basis $\mathscr{B}_1 \cup \mathscr{F}$ is denoted by $\mathscr{B}_1(\mathscr{F})$.

Definition 2 (*AC* hierarchy) *A language $X \subseteq \{0,1\}^*$ belongs to the class $AC^i(\mathscr{F})$ iff X is accepted by a polynomial size, $\log^i$-depth family of unbounded fan-in Boolean circuits over the basis $\mathscr{B}_1(\mathscr{F})$. AC^i is defined to be $AC^i(\emptyset)$.*

It is not hard to see that

$$AC^i \subseteq AC^i(\text{MAJ}) \subseteq AC^i(st\text{CONN}_2) \subseteq AC^i(st\text{CONN}) \subseteq AC^{i+1}.$$

In the meantime, it is known that the union $\bigcup_i AC^i$ coincides with the class *NC*, i.e., the class of those languages for which there is a polynomial size, polylog-depth family of bounded fan-in circuits. The class *P/poly*, often called *nonuniform P*, consists of those languages which are computable in polynomial time with polynomial advice. It is also known that *P/poly* coincides with the class of languages for which there is a polynomial size family of Boolean circuits. There are some separation results for constant depth circuits. For instance, it is known that $AC^i \subsetneq AC^i(\text{PARITY}) \subsetneq AC^i(\text{MAJ})$ for $i = 0$ [3, 14], though no such results are known for $i > 0$.

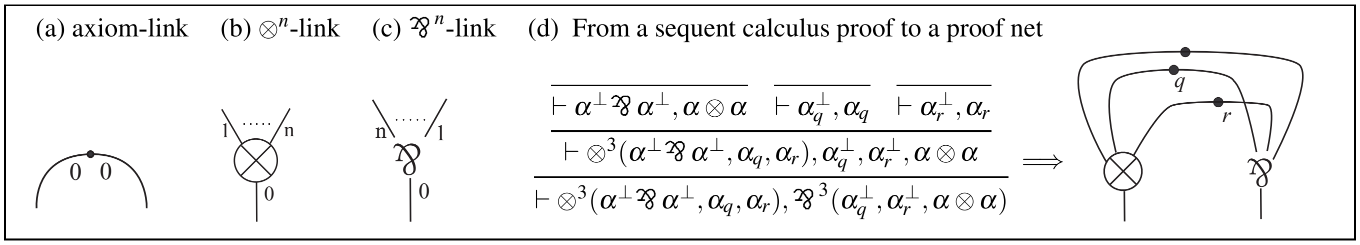

(a) axiom-link
(b) $\otimes^n$-link
(c) $\parr^n$-link
(d) From a sequent calculus proof to a proof net
0 0
1 ..... n
0
n 1
0
$\vdash \alpha^\perp \parr \alpha^\perp, \alpha \otimes \alpha$
$\vdash \alpha_q^\perp, \alpha_q$
$\vdash \alpha_r^\perp, \alpha_r$
$\vdash \otimes^3(\alpha^\perp \parr \alpha^\perp, \alpha_q, \alpha_r), \alpha_q^\perp, \alpha_r^\perp, \alpha \otimes \alpha$
$\vdash \otimes^3(\alpha^\perp \parr \alpha^\perp, \alpha_q, \alpha_r), \parr^3(\alpha_q^\perp, \alpha_r^\perp, \alpha \otimes \alpha)$
$\Longrightarrow$
q
r

Figure 1. Links and an example of proof net construction

3. Proof Nets for Unbounded Fan-in MLL

3.1. Sequent Calculus

In this section, we introduce an unbounded fan-in version of multiplicative linear logic **MLL**, denoted by **MLLu**. **MLLu** is like **MLL**, but equipped with logical connectives of arbitrary arities. It should be noted that there is no difference between **MLL** and **MLLu** as to representability of functions; **MLLu** just gives us a depth-efficient way of writing proofs (such a system with generalized multiplicatives is studied in [2]).

Definition 3 (Formulas of MLLu) *The* formulas *A, B, $C, \ldots$ of* **MLLu** *are built from literals α, $\alpha^\perp$, β, $\beta^\perp, \ldots$ by n-ary versions of* multiplicative conjunction $\otimes^n(A_1,\ldots,A_n)$ *and* multiplicative disjunction $\parr^n(A_1,\ldots,A_n)$ *for every $n \geq 1$.*

The *negation* $A^\perp$ of a non-literal formula A is defined by de Morgan duality:

$$\begin{array}{rcl}(\otimes^n(A_1,\ldots,A_n))^\perp & \equiv & \parr^n(A_n^\perp,\ldots,A_1^\perp)\\ (\parr^n(A_1,\ldots,A_n))^\perp & \equiv & \otimes^n(A_n^\perp,\ldots,A_1^\perp)\end{array}$$

Note that the order of subformulas is reversed by negation (as in non-commutative linear logic). The notation $A[B/\alpha]$ denotes the formula A with all occurrences of α replaced by B. In the sequel, we use the following abbreviations:

$$\begin{array}{rclrcl}A \otimes B & \equiv & \otimes^2(A,B) & A \parr B & \equiv & \parr^2(A,B)\\ A \multimap B & \equiv & A^\perp \parr B & A^n & \equiv & \otimes^n(A,\ldots,A)\end{array}$$

Superscripts n in $\otimes^n$ and $\parr^n$ are often omitted. In addition, a sequence $A_1,\ldots,A_n$ is written as $\overrightarrow{A}$, and $A_n,\ldots,A_1$ as $\overleftarrow{A}$.

Definition 4 (Sequent calculus for MLLu) *A* sequent *of* **MLLu** *is of the form $\vdash \Gamma$, where Γ is a multiset of formulas. The* inference rules *of* **MLLu** *are as follows:*

$$\frac{}{\vdash A, A^\perp}\ (Axiom) \qquad \frac{\vdash \Gamma_1, A_1 \quad \cdots \quad \vdash \Gamma_n, A_n}{\vdash \Gamma_1,\ldots,\Gamma_n, \otimes^n(\overrightarrow{A})}\ \otimes^n$$

$$\frac{\vdash \Gamma, C \quad \vdash \Delta, C^\perp}{\vdash \Gamma, \Delta}\ (Cut) \qquad \frac{\vdash \Gamma, A_n, \ldots, A_1}{\vdash \Gamma, \parr^n(\overleftarrow{A})}\ \parr^n$$

Thus **MLLu** has neither weakening nor contraction, while it admits exchange implicitly. The formulas C and $C^\perp$ in the rule (Cut) are called *cut formulas*.

3.2. Proof Nets

Let us now introduce the proof nets [4, 2] (see also [10] for an excellent exposition). Informally, a proof net is a graphical structure which is obtained from a sequent calculus proof by abstracting away everything irrelevant to computation. Proof nets just keep the structure of proofs.

The basic ingredients are three sorts of *link* (see Figure 1(a), (b) and (c)). These are called an axiom link, $\otimes^n$-link and $\parr^n$-link, respectively. Each link has several *ports* to which distinct natural numbers are associated, with the exception that an axiom link has two ports both numbered by 0. The port(s) numbered by 0 is called the *principal port(s)*, while others are called *auxiliary ports*. By convention, the principal port(s) is always written below a link, while auxiliary ports are above it. The auxiliary ports are ordered from left to right for a $\otimes^n$-link, while they are written in the reverse order for a $\parr^n$-link. With this convention, the port numbers can be safely omitted. Note that, unlike the standard formulation, there is no link for (Cut). It is rather represented by a pair of links connected to each other at the principal ports.

A proof net is obtained from a sequent calculus proof by replacing the inference rules (other than (Cut)) with the corresponding links and connecting them by edges corresponding to the subformula relation and (Cut) inferences. Figure 1(d) illustrates how to obtain a proof net from a sequent calculus proof (where subscripts q and r are attached to formulas/links in order to distinguish occurrences).

The formal definition is as follows.

Definition 5 (Pseudo nets) *A* pseudo net *P is a triple $< L, \sigma, \sim >$ such that:*

- *L is a finite set of* links*;*
- *$\sigma : L \longrightarrow \{\bullet\} \cup \{\otimes^n, \parr^n\}_{n \geq 1}$;*
- *$\sim$ is a symmetric relation on $(L \times \mathbb{N})$.*

(a) Type inference rules

$$\frac{}{\vdash p : A, p : A^{\perp} \rhd \mathsf{ax}_p}\ (Axiom) \qquad \frac{\vdash \Gamma_1, p_1 : A_1 \rhd P_1 \quad \cdots \quad \vdash \Gamma_n, p_n : A_n \rhd P_n}{\vdash \Gamma_1, \ldots, \Gamma_n, q : \otimes^n(\overrightarrow{A}) \rhd \mathsf{tensor}_q^{\overrightarrow{p}}(\vec{P})}\ \otimes^n$$

$$\frac{\vdash \Gamma, p : C \rhd P \quad \vdash \Delta, q : C^{\perp} \rhd Q}{\vdash \Gamma, \Delta \rhd \mathsf{cut}^{p,q}(P,Q)}\ (Cut) \qquad \frac{\vdash \Gamma, p_n : A_n, \ldots, p_1 : A_1 \rhd P}{\vdash \Gamma, q : ⅋^n(\overleftarrow{A}) \rhd \mathsf{par}_q^{\overleftarrow{p}}(P)}\ ⅋^n$$

(b) $\mathsf{tensor}_q^{p_1,\ldots,p_n}(P_1,\ldots,P_n)$ (c) $\mathsf{par}_q^{p_n,\ldots,p_1}(P)$ (d) $\mathsf{cut}^{p,q}(P,Q)$

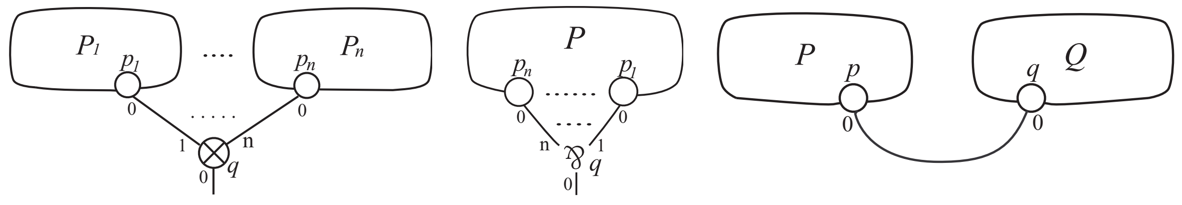


Figure 2. Type inference rules and proof net constructors

A link p with $\sigma(p) = \bullet$ ($⅋^n$, $\otimes^n$, resp.) stands for an *axiom-link* (*$⅋^n$-link*, *$\otimes^n$-link*, resp.). When $(p,n) \sim (q,m)$, we say that there is an *edge* between (p,n) and (q,m), where (p,n) stands for a *port* of link p numbered by n. A *cut* in P is an unordered pair of links p,q such that $(p,0) \sim (q,0)$. A cut $\{p,q\}$ is called an *a-cut* when either p or q is an axiom-link; otherwise it is called an *m-cut*.

Definition 6 (Proof nets) *A* judgment *is of the form* $\vdash \Gamma \rhd P$*, where P is a pseudo net and Γ is a multiset of expressions of the form $p : A$. A* proof net of type $\vdash \Gamma$ *is a pseudo net P such that $\vdash \Gamma \rhd P$ is derivable by the type inference rules in Figure 2(a), where:*

- ax_p *denotes the pseudo net which consists of a single axiom link p with no edges;*
- $\mathsf{tensor}_q^{p_1,\ldots,p_n}(P_1,\ldots,P_n)$ *denotes the pseudo net which extends the disjoint union of $P_1,\ldots,P_n$ with a new $\otimes^n$-link q and a new edge $(p_i,0) \sim (q,i)$ for each $1 \le i \le n$ (Figure 2(b));*
- $\mathsf{par}_q^{p_n,\ldots,p_1}(P)$ *denotes the pseudo net which extends P with a new $⅋^n$-link q and a new edge $(p_i,0) \sim (q,i)$ for each $1 \le i \le n$ (Figure 2(c));*
- $\mathsf{cut}^{p,q}(P,Q)$ *denotes the pseudo net which extends the disjoint union of P and Q with a new edge $(p,0) \sim (q,0)$ (see Figure 2(d)).*

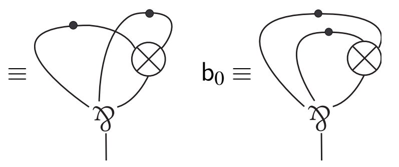
A proof net of type $\vdash p : A$ for a single formula A is simply called a proof net of type A.

Note that cuts are only created by the (Cut) rule, and it is ensured that if $\{p,q\}$ is an m-cut and p is a $\otimes^n$-link ($⅋^n$-link, resp.), then q is a $⅋^n$-link ($\otimes^n$-link, resp.). It is clear that every sequent calculus proof π induces a unique proof net, whereas the converse does not hold. Indeed, whenever we have $\vdash \Gamma \rhd P$, we also have $\vdash \Gamma[A/\alpha] \rhd P$ for every A and α.

As an example, consider how to represent Boolean values by proof nets. In simply typed lambda calculus, Boolean values are represented by terms of type $\alpha \to \alpha \to \alpha$. However, we cannot use its analogue $\alpha \multimap \alpha \multimap \alpha$, because it does not have any proofs due to lack of weakening. Instead, the Boolean type $\mathbf{B}$ is defined as $⅋^3(\alpha^{\perp}, \alpha^{\perp}, \alpha \otimes \alpha)$. There are exactly two cut-free proof nets (up to renaming of links) of this type:

$$\begin{aligned} \mathsf{b}_1 &\equiv \mathsf{par}_s^{p,q,r}(\mathsf{tensor}_r^{p,q}(\mathsf{ax}_p, \mathsf{ax}_q)) \\ \mathsf{b}_0 &\equiv \mathsf{par}_s^{q,p,r}(\mathsf{tensor}_r^{p,q}(\mathsf{ax}_p, \mathsf{ax}_q)) \end{aligned}$$

depicted as:



They serve as *true* and *false*, respectively. From a graph theoretic point of view, the difference between b_1 and b_0 amounts to the fact that b_1 is not planar whereas b_0 is.

Definition 7 (Depth and size) *The* depth $d(A)$ *of a formula A is given by $d(\alpha) = d(\alpha^{\perp}) = 1$ and $d(\otimes^n(\overrightarrow{A})) = d(⅋^n(\overleftarrow{A_1})) = max(d(A_1),\ldots,d(A_n)) + 1$.*

Given a derivation π of $\vdash \Gamma \rhd P$, its depth $d(\pi)$ *is the maximum depth of cut formulas in it. We also define the* total depth $d'(\pi)$ *to be the maximum depth of cut formulas and all formulas in* Γ.

The depth $d(P)$ *of a proof net P is defined to be*

$$min\{d(\pi) | \pi \text{ is a derivation of} \vdash \Gamma \rhd P \text{ for some } \Gamma\}.$$

The total depth $d'(P)$ *is defined analogously. The* size $|P|$ *is the number of links in P.*

Observe that $d(A[B/\alpha]) \leq d(A) + d(B) - 1$, $d(P) \leq d'(P)$, and $d'(\mathsf{cut}^{p,q}(P,Q)) \leq max(d'(P), d'(Q))$.

Remark. It is possible to consider the notion of *principal derivation*, that is to say the most general derivation for a given proof net from which any other derivations are obtained by substitution and permutation of inferences. Then the above $d(P)$ amounts to the depth of the principal derivation for P.

3.3. Cut-Elimination

Definition 8 (A-reduction and m-reduction) *Given a proof net P, an* a-reduction *consists in replacing an a-cut* $\{p,q\}$ *and edges incident to* p,q *as in Figure 3(a). A m-reduction consists in replacing a m-cut* $\{p,q\}$ *and edges incident to* p,q *as in Figure 3(b).*

We write $P \longrightarrow Q$ when Q is obtained from P either by an a-reduction or an m-reduction. The relation $\longrightarrow^*$ is defined to be the transitive reflexive closure of $\longrightarrow$. Note that the number of links strictly decreases by a reduction, due to our convention that cuts are expressed by edges rather than links.

To give an example of cut-elimination, consider again the proof net given in Figure 1(d), which is hereafter denoted by $\mathsf{not}(p)$. It is of type $\vdash p : \mathbf{B}^\perp, s : \mathbf{B}$. Link p is naturally considered as an input argument, while link q is considered as an output argument. Now compose it with b_1 at p by (Cut) to obtain a proof net $\mathsf{not}(\mathsf{b}_1)$, that is of type $\mathbf{B}$. We then have $\mathsf{not}(\mathsf{b}_1) \longrightarrow^* \mathsf{b}_0$ as in Figure 3(c). Similarly, one can check that $\mathsf{not}(\mathsf{b}_0) \longrightarrow^* \mathsf{b}_1$. Therefore, $\mathsf{not}(p)$ can be seen as representing the negation function, in such a way that when an input Boolean value is given at a link of type $\mathbf{B}^\perp$ by (Cut), the output is obtained via cut-elimination.

The following facts are fundamental [4]:

Theorem 9 *The following holds for every proof net P:*

- *Sequential cut-elimination: P reduces to a cut-free proof net* P_0 *within* $|P|$ *reduction steps, where* P_0 *is unique up to renaming of links.*

- *Subject reduction: If* $\vdash \Gamma \rhd P$ *and* $P \longrightarrow^* Q$*, then* $\vdash \Gamma \rhd Q$.

4. From Boolean Circuits to Proof Nets

4.1. Flat Boolean Proof Nets

In this section, we give a depth-efficient simulation of Boolean circuits by proof nets. Before considering the general case, let us first consider a simple situation.

Definition 10 (Flat Boolean proof nets) *A* flat Boolean proof net with n inputs *is a proof net* $P(\overrightarrow{p})$ *of type*

$$\vdash p_1 : \boldsymbol{B}^\perp, \ldots, p_n : \boldsymbol{B}^\perp, q : \boldsymbol{B}.$$

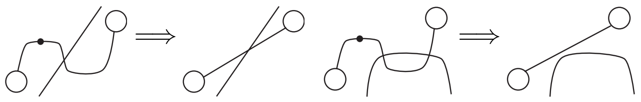
Such a proof net has *n input links* $\overrightarrow{p} \equiv p_1, \ldots, p_n$ and one *output link q* (we often do not indicate the output link). It is called *flat* because the input types and the output type are of the same depth. Given $\overrightarrow{\mathsf{b}} \equiv \mathsf{b}_{i_1}, \ldots, \mathsf{b}_{i_n}$, $P(\overrightarrow{\mathsf{b}})$ denotes the proof net obtained by connecting b_i to p_i by (Cut) for every $1 \leq i \leq n$ (see Figure 3(d)). The resulting net $P(\overrightarrow{\mathsf{b}})$ is of type $\mathbf{B}$, and by Theorem 9, it reduces to a unique cut-free proof net of type $\mathbf{B}$, that is either b_1 or b_0. We say that $P(\overrightarrow{p})$ *represents* a function $f : \{0,1\}^n \longrightarrow \{0,1\}$ if $P(\mathsf{b}_{i_1}, \ldots, \mathsf{b}_{i_n}) \longrightarrow^* \mathsf{b}_{f(w)}$ for every $w = i_1 \cdots i_n \in \{0,1\}^n$.

Parity. We have already seen that the proof net $\mathsf{not}(p)$ in Figure 1(d) represents the negation function. As another example, we have a proof net $\mathsf{parity}^n(\overrightarrow{p})$ representing PARITY^n for every n (see Figure 4(a)). Although the correctness of $\mathsf{parity}^n(\overrightarrow{p})$ can be checked directly, one might like to employ the following graph theoretic argument. Below, an *a-edge* means an edge incident to an axiom-link.

Let $i_1 \cdots i_n \in \{0,1\}^n$, and suppose that the sum of $i_1, \ldots, i_n$ is k. Let $\overrightarrow{\mathsf{b}} \equiv \mathsf{b}_{i_1}, \ldots, \mathsf{b}_{i_n}$ and consider the proof net $\mathsf{parity}(\overrightarrow{\mathsf{b}})$ of type $\mathbf{B}$. Now, our argument goes as follows; first, it is clear that $\mathsf{parity}(\overrightarrow{\mathsf{b}})$ has a drawing with exactly k crossings, one for each drawing of b_1. Note that those crossings are all between a-edges. Second, thinking of the reduction rules in Figure 3 (a) and (b) as rewrite rules for *drawings* of proof nets, we may observe:

- An m-reduction does not change the number of crossings (since there are no crossings at m-cuts).

- An a-reduction does not change the *parity* of the number of crossings:



As a consequence, what we obtain after cut-elimination is (a drawing of) a cut-free proof net of type $\mathbf{B}$ with the parity of crossings equivalent to $k \mod 2$. That is nothing but $\mathsf{b}_{k \bmod 2}$, as required.

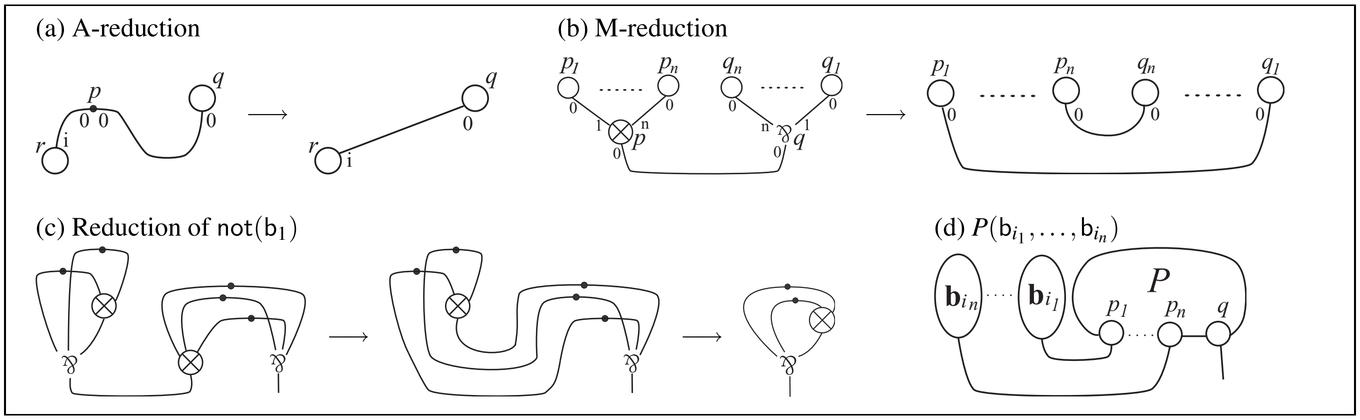
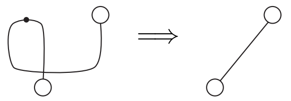


Figure 3. Reductions

Actually, the above argument would break down if there was a *self-crossing*:



However, we can show that such a self-crossing never appears in the current setting.

From the above argument, it is clear that if one adds a twist somewhere between a-edges in parity($\overrightarrow{p}$), it will result in a proof net for $\neg$PARITY^n. Furthermore, the above argument applies to *any* cut-free flat Boolean proof net (when suitably drawn). That is to say, all what matters is the *parity* of the number of crossings. We therefore conclude:

Theorem 11 *Every flat Boolean proof net with n inputs represents either* PARITY^n *or* $\neg$PARITY^n.

A consequence is that we need to consider a more general class of proof nets than flat ones to represent arbitrary Boolean functions.

4.2. General Case

We generalize the flat Boolean proof nets in two ways. First, we allow an input type to be of the form $\mathbf{B}^\perp[A]$ with A arbitrary, where $\mathbf{B}^\perp[A]$ is short for $\mathbf{B}^\perp[A/\alpha]$. Note that b_1 and b_0 have type $\mathbf{B}[A]$ for arbitrary A, hence $\mathbf{B}^\perp[A]$ can still be considered as an input type. Second, we allow an output type to be accompanied by some *garbage* $\overrightarrow{C}$.

Definition 12 (Boolean proof nets) *A* Boolean proof net with n inputs *is a proof net* $P(\overrightarrow{p})$ *of type*

$$\vdash p_1 : \boldsymbol{B}^\perp[A_1], \ldots, p_n : \boldsymbol{B}^\perp[A_n], q : \otimes^{m+1}(\boldsymbol{B}, \overrightarrow{C})$$

for some $\overrightarrow{A} \equiv A_1, \ldots, A_n$ *and* $\overrightarrow{C} \equiv C_1, \ldots, C_m$.

Given $\overrightarrow{\mathsf{b}} \equiv \mathsf{b}_{i_1}, \ldots, \mathsf{b}_{i_n}$, $P(\overrightarrow{\mathsf{b}})$ is defined as before (Figure 3(d)), but this time its type is $\otimes^{m+1}(\mathbf{B}, \overrightarrow{C})$. It is easy to see that $P(\overrightarrow{\mathsf{b}})$ reduces to a cut-free proof net of the form $\mathsf{tensor}(\mathsf{b}_i, \overrightarrow{Q})$ with $i \in \{0,1\}$. In this case, we say that $P(\overrightarrow{\mathsf{b}})$ *evaluates* to b_i and write $P(\overrightarrow{\mathsf{b}}) \xrightarrow{\mathbf{ev}} \mathsf{b}_i$.

An n-ary Boolean proof net $P(\overrightarrow{p})$ *represents* a function $f : \{0,1\}^n \longrightarrow \{0,1\}$ if $P(\mathsf{b}_{i_1}, \ldots, \mathsf{b}_{i_n})$ evaluates to $\mathsf{b}_{f(w)}$ for every $w \equiv i_1 \cdots i_n \in \{0,1\}^n$. In this case, we also say that $P(\overrightarrow{p})$ *accepts* the language X given by $X = f^{-1}(1)$.

In the sequel, we describe several constructions of Boolean proof nets.

Conditional. Given two proof nets P_1 and P_2 of type $\vdash \Gamma, p_1 : A$ and $\vdash \Delta, p_2 : A$, one builds a proof net $\mathsf{cond}_r^{p_1,p_2}[P_1,P_2](q)$ in Figure 4(b). It is of type

$$\vdash \Gamma, \Delta, q : \mathbf{B}[A]^\perp, r : A \otimes A.$$

Given an Boolean input b_i at q, it reduces as follows:

$$\begin{array}{rcl} \mathsf{cond}_r^{p_1,p_2}[P_1,P_2](\mathsf{b}_1) & \longrightarrow^* & \mathsf{tensor}_r^{p_1,p_2}(P_1,P_2) \\ \mathsf{cond}_r^{p_1,p_2}[P_1,P_2](\mathsf{b}_0) & \longrightarrow^* & \mathsf{tensor}_r^{p_2,p_1}(P_2,P_1) \end{array}$$

With the convention that the first component is considered as the real output, cond surely works as conditional. We omit scripts p_1, p_2, r when they are clear from the context.

Disjunction. Define a proof net for binary disjunction by

$$\mathsf{or}(p_1,p_2) \equiv \mathsf{cond}[\mathsf{b}_1, \mathsf{ax}_{p_1}](p_2)$$

which is of type $\vdash p_1 : \mathbf{B}^\perp, p_2 : \mathbf{B}^\perp[\mathbf{B}], q : \mathbf{B} \otimes \mathbf{B}$. It surely represents disjunction, as follows:

$$\begin{array}{l} \mathsf{or}(\mathsf{b}_i,\mathsf{b}_1) \longrightarrow^* \mathsf{cond}[\mathsf{b}_1,\mathsf{b}_i](\mathsf{b}_1) \longrightarrow^* \mathsf{tensor}(\mathsf{b}_1,\mathsf{b}_i) \\ \mathsf{or}(\mathsf{b}_i,\mathsf{b}_0) \longrightarrow^* \mathsf{cond}[\mathsf{b}_1,\mathsf{b}_i](\mathsf{b}_0) \longrightarrow^* \mathsf{tensor}(\mathsf{b}_i,\mathsf{b}_1) \end{array}$$

It is also possible to define n-ary disjunction for every n, by *composing* binary disjunctions.

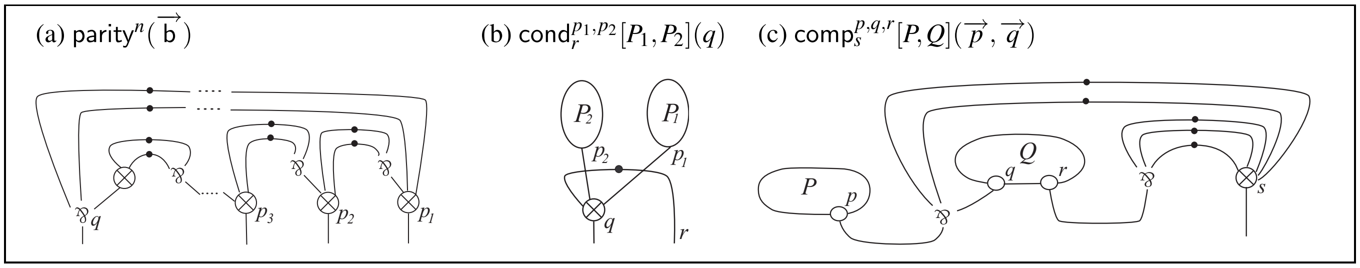

(a) $\mathsf{parity}^n(\overrightarrow{\mathsf{b}})$
(b) $\mathsf{cond}_r^{p_1,p_2}[P_1,P_2](q)$
(c) $\mathsf{comp}_s^{p,q,r}[P,Q](\overrightarrow{p},\overrightarrow{q})$
q
p_3
p_2
p_1
P_2
P_1
p_2
p_1
q
r
P
p
Q
q
r
s

Figure 4. Parity, Conditional and Composition

Lemma 13 (Composition) *Let $\Gamma \equiv p'_1 : A'_1, \dots, p'_n : A'_n$ and $\Delta \equiv q'_1 : B'_1, \dots, q'_n : B'_m$. Given proof nets $P(\overrightarrow{p'})$ and $Q(q,\overrightarrow{q'})$ of type*

$$\vdash \Gamma, p : \otimes(\boldsymbol{B}, \overrightarrow{C}) \text{ and } \vdash q : \boldsymbol{B}^{\perp}[A], \Delta, r : \otimes(\boldsymbol{B}, \overrightarrow{D}),$$

there is a proof net $\mathsf{comp}_s^{p,q,r}[P,Q](\overrightarrow{p'},\overrightarrow{q'})$ of type

$$\vdash \Gamma[A], \Delta, s : \otimes(\boldsymbol{B}, \overrightarrow{D}, \overrightarrow{C[A]})$$

such that the total depth is bounded by $max(d'(P) + d(A) - 1, d'(Q))$ and whenever $P(\overrightarrow{R}) \xrightarrow{\mathbf{ev}} S$ and $Q(S, \overrightarrow{T}) \xrightarrow{\mathbf{ev}} U$, we have $\mathsf{comp}_s^{p,q,r}[P,Q](\overrightarrow{R},\overrightarrow{T}) \xrightarrow{\mathbf{ev}} U$.

Proof. The type of P can be lifted to $\vdash \Gamma[A], p : \otimes(\mathbf{B}[A], \overrightarrow{C[A]})$. Observe that all formulas in it have depth bounded by $d'(P) + d(A) - 1$. From this and the type of Q, one can easily derive:

$$\dfrac{\vdash \Gamma[A], \otimes(\mathbf{B}[A], \overrightarrow{C[A]}) \quad \vdash \mathbf{B}^{\perp}[A], \Delta, \otimes(\mathbf{B}, \overrightarrow{D})}{\vdash \Gamma[A], \Delta, \otimes(\mathbf{B}, \overrightarrow{D}, \overrightarrow{C[A]})}$$

The derivation yields the proof net given in Figure 4(c), which surely works as composition of P and Q. Since all the formulas occurring in the above derivation have depth bounded by the depths of those in the premises, we conclude that the total depth is bounded by $max(d'(P) + d(A) - 1, d'(Q))$. ■

Coming back to disjunction, apply the previous lemma to

$$\vdash p_1 : \mathbf{B}^{\perp}, p_2 : \mathbf{B}^{\perp}[\mathbf{B}], p : \mathbf{B} \otimes \mathbf{B} \rhd \mathsf{or}(p_1, p_2)$$
$$\vdash q : \mathbf{B}^{\perp}, p_3 : \mathbf{B}^{\perp}[\mathbf{B}], r : \mathbf{B} \otimes \mathbf{B} \rhd \mathsf{or}(q, p_3)$$

to obtain $\mathsf{or}^3(p_1, p_2, p_3)$ defined as

$$\mathsf{comp}_s^{p,q,r}[\mathsf{or}(p_1,p_2), \mathsf{or}(q,p_3)](p_1,p_2,p_3).$$

It is of type $\vdash p_1 : \mathbf{B}^{\perp}, p_2 : \mathbf{B}^{\perp}[\mathbf{B}], p_3 : \mathbf{B}^{\perp}[\mathbf{B}], s : \otimes(\mathbf{B},\mathbf{B},\mathbf{B})$. By repetition, one can obtain n-ary disjunction for every n. Note that the total depth is $d(\mathbf{B}^{\perp}[\mathbf{B}])$, that is a constant independent of n.

Duplication. Let $n \geq 2$ and $C \equiv \otimes^n(\mathbf{B}[A_1], \dots, \mathbf{B}[A_n])$. Define $\mathsf{copy}^n(p) \equiv \mathsf{cond}[\mathsf{tensor}(\overrightarrow{\mathsf{b}_1}), \mathsf{tensor}(\overrightarrow{\mathsf{b}_0})](p)$, which is of type $\vdash p : \mathbf{B}^{\perp}[C], q : C \otimes C$. It produces n copies of the input Boolean value, one for each type $\mathbf{B}[A_i]$, as follows:

$$\mathsf{copy}^n(\mathsf{b}_1) \longrightarrow^* \mathsf{tensor}(\mathsf{tensor}(\overrightarrow{\mathsf{b}_1}), \mathsf{tensor}(\overrightarrow{\mathsf{b}_0}))$$
$$\mathsf{copy}^n(\mathsf{b}_0) \longrightarrow^* \mathsf{tensor}(\mathsf{tensor}(\overrightarrow{\mathsf{b}_0}), \mathsf{tensor}(\overrightarrow{\mathsf{b}_1}))$$

This proof net is useful for encoding a gate of arbitrary fan-out. The total depth is $d(\mathbf{B}[C])$, which only depends on $max(d(A_1), \dots, d(A_n))$.

Majority. Let us consider MAJ. The encoding is particularly interesting, because it makes use of higher order facility of proof nets. We first illustrate the idea.

Assume for simplicity that $n = 2m$. Let id be the identity function on $\{0,1\}^{n+1}$, and sh be the shift function on $\{0,1\}^{n+1}$, given by $sh(i_1 \cdots i_{n+1}) = i_2 \cdots i_{n+1} i_1$ for every $i_1 \cdots i_{n+1} \in \{0,1\}^{n+1}$. Now, let F be a *higher order functional* whose output is a function, defined by $F(0) = id$ and $F(1) = sh$.

Using this F, $\text{MAJ}^n(x_1 \cdots x_n)$ can be defined as

$$FirstBit(F(x_1) \circ \cdots \circ F(x_n)(\underbrace{0 \cdots 0}_{m}\underbrace{1 \cdots 1}_{m+1})),$$

where $FirstBit(x_1 \cdots x_{n+1}) = x_1$ and $\circ$ stands for functional composition. This definition is correct; for instance, we have:
$\text{MAJ}^6(101101)$
$= FirstBit(F(1)F(0)F(1)F(1)F(0)F(1)(0001111))$
$= FirstBit(sh \circ id \circ sh \circ sh \circ id \circ sh(0001111))$
$= FirstBit(1110001) = 1$.

One can easily encode id and sh as proof nets of type $\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}$. Hence by conditional, the proof net $F(p)$ for the functional F can be defined as $\mathsf{cond}[sh, id](p)$ of type

$$\vdash p : \mathbf{B}^{\perp}[\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}], r : (\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}) \otimes (\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}).$$

The output of $F(p_i)$ is a pair of (proof nets coding) *id* and *sh*, connected by tensor. The first is useful whereas the second is garbage. Now apply the outputs of $F(p_1),\ldots,F(p_n)$ to $(\mathsf{tensor}(\overrightarrow{\mathsf{b}_I}),\mathsf{tensor}(\overrightarrow{\mathsf{b}_G}))$ componentwise, where $\overrightarrow{\mathsf{b}_I} \equiv \underbrace{\mathsf{b}_0,\ldots,\mathsf{b}_0}_{m\ times},\underbrace{\mathsf{b}_1,\ldots,\mathsf{b}_1}_{m+1\ times}$ and $\overrightarrow{\mathsf{b}_G}$ is arbitrary. The resulting net is of type

$$\vdash \overrightarrow{p}:\overrightarrow{\mathbf{B}^{\perp}[\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}]}, r:\mathbf{B}^{n+1}\otimes\mathbf{B}^{n+1},$$

with $\overrightarrow{p} \equiv p_1,\ldots,p_n$. It is then easy to extract the first bit. The result is surely a Boolean proof net. The size is $O(n^2)$, and the total depth is $d(\mathbf{B}[\mathbf{B}^{n+1} \multimap \mathbf{B}^{n+1}])$, that is a constant independent of n.

St-connectivity for undirected graphs of degree 2. Suppose that $E = \{x_{i,j}\}_{1\le i<j\le n}$ codes an undirected graph G of degree 2. E consists of $m = n(n-1)/2$ Boolean values, and $st\textsc{conn}_2^n(E) = 1$ holds iff there is a path from vertex 1 to n in G. To check the latter, we consider a token travelling around the graph, starting from vertex 1. The location of the token is specified by an element of $\{0,1\}^n$ such that the ith bit is 1 iff the token is currently visiting vertex i. To simulate a transition of a token, we exploit the function $sw_{i,j} : \{0,1\}^n \longrightarrow \{0,1\}^n$ which swaps the ith bit with the jth.

Given $k \ge 0$, let $x_{i,j}$ be the $(k \bmod m)$th element of E. Define a functional F by $F(k) = sw_{i,j}$ if $x_{i,j} = 1$ and $F(k) = id$ otherwise, and let

$$H(k) = LastBit(F(k)\circ\cdots\circ F(1)(1\underbrace{0\cdots0}_{n-1})).$$

It is then clear that the token, initially located at vertex 1, reaches vertex n whenever $H(k) = 1$ for some $k \ge 0$, because an application of function $F(l)$ $(1 \le l \le k)$ corresponds to a possible transition of the token. Conversely, whenever there is a path form vertex 1 to n, there is some $k \le n\cdot m$ such that $H(k) = 1$. The reason is that the token moves to an adjacent vertex (if any) in at most m steps of applications of $F(l)$, and furthermore, when the current vertex visited by the token is of degree 2, the token will never go back to the previous one, because it will find an edge leading to the other vertex first. Since there are only n vertices, the token will reach vertex n in $n\cdot m$ steps whenever possible. We therefore have $st\textsc{conn}_2^n(E) = \bigvee_{k\le n\cdot m} H(k)$.

As before, one can implement $st\textsc{conn}_2^n$ by a proof net of constant depth; the crucial fact here is that both *id* and $sw_{i,j}$ are represented by a *flat* proof net of type $\mathbf{B}^n \multimap \mathbf{B}^n$. The size is $O(m^4)$.

Remark. The same idea would work for acyclic graphs of arbitrary degree too, but fails for graphs in general. Indeed, it would be a great surprise if there was a constant depth proof net for $st\textsc{conn}$, because that would imply $st\textsc{conn} \in AC^0(st\textsc{conn}_2)$ by Theorem 21.

From what precedes, we conclude:

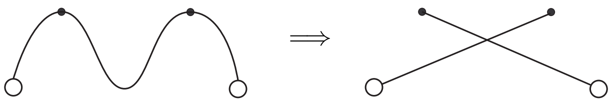
Theorem 14 *For every unbounded fan-in Boolean circuit C of size s and depth d over the basis $\mathscr{B}_1(st\textsc{conn}_2)$, there is a Boolean proof net of size $O(s^5)$ and depth $O(d)$ which accepts the same set as C does.*

Proof. Every gate of fan-in n and fan-out m can be encoded by a proof net of size $O(n^4+m) \le O(s^4)$ and of constant depth. The depth increases when one composes an encoding of a gate with another by using Lemma 13. The increase is linear in d. ∎

5. From Proof Nets to Boolean Circuits

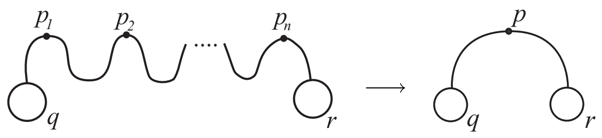
5.1. Parallel Cut Elimination

Recall that Theorem 9 is concerned with a sequential cut-elimination procedure. As a result, the number of reduction steps required is linear in the size of the proof net, that is too slow from the viewpoint of parallel computation. Here we present a parallel cut-elimination procedure to achieve a speed-up. Although m-reductions are not problematic at all, there is a small problem for a-reductions. The problem is that there are two ways of reducing an a-cut between two axioms, and applying both ways in parallel causes a conflict:



This conflict is avoided in [10] by thinking of axioms not as links but as *wires*. We cannot, however, use the same trick, because wires are difficult to implement by circuits. Instead, we introduce another reduction step which is global and eliminates several a-cuts at once.

Definition 15 (Tightening reduction) *We presuppose that every proof net is endowed with a total ordering $\prec$ on the links. An* a-sequence *is a sequence of axiom-links $p_1,\ldots,p_n$ $(n \ge 2)$ such that $(p_i,0) \sim (p_{i+1},0)$ for every $1 \le i \le n-1$. Such a sequence is* maximal *if it cannot be extended in either direction. A* t-reduction *(tightening reduction) consists in replacing a maximal a-sequence as follows:*



where p is the smaller one of p_1 and p_n with respect to $\prec$.

Definition 16 (Parallel reduction) *We write $P \Longrightarrow_a Q$ ($P \Longrightarrow_m Q$, $P \Longrightarrow_t Q$, resp.) when Q is obtained from P by applying a-reductions (m-reductions, t-reductions, resp.) to all a-cuts (m-cuts, maximal a-sequences, resp.) in P simultaneously. We write $P \Longrightarrow Q$ if $P \Longrightarrow_a Q$, $P \Longrightarrow_m Q$ or $P \Longrightarrow_t Q$.*

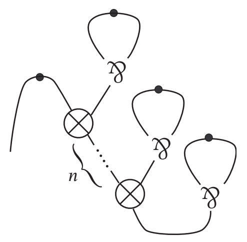
Theorem 17 (Parallel cut-elimination) *There is a sequence of parallel reductions*

$$P \Longrightarrow P_1 \Longrightarrow P_2 \cdots \Longrightarrow P_n$$

such that P_n is cut-free and $n \leq 3 \cdot d(P)$.

Proof. We show that one can decrease the depth by 1 in three steps. Given P, first apply $\Longrightarrow_t$ to obtain P_1. In P_1, there is no pair of axiom links connected each other. Hence one can safely apply $\Longrightarrow_a$ to obtain P_2 without any conflicts. Note that P_2 contains only m-cuts. Finally apply $\Longrightarrow_m$ to obtain P_3. We then have $d(P_2) > d(P_3)$, because, assuming a typing derivation for P_2, it replaces each cut of type $(\otimes^n(\overrightarrow{A}), ⅋^n(\overleftarrow{A^\perp}))$ with cuts of type $(A_1, A_1^\perp), \ldots, (A_n, A_n^\perp)$. ■

Remark. It is not always the case, however, that parallelization achieves a speed-up. For instance, consider the following proof net P_n (which corresponds to $\underbrace{I \cdots I}_{n\ times}$ with $I \equiv \lambda x.x$ in lambda calculus):



It is not hard to see that both $|P_n|$ and $d(P_n)$ are linear in n. This means that for such a proof net, parallel cut-elimination requires of essentially the same time as sequential cut-elimination. This shows a limitation on parallelization.

5.2. Simulating Cut-Elimination

Let us now give a simulation of parallel cut-elimination by Boolean circuits. To do so, we first need to represent each proof net by a set of Boolean values. In the sequel, we fix a set L_0 of links, and only consider the proof nets with links from L_0.

Definition 18 (Configurations) *A* configuration θ *consists of the following Boolean values:*

- *$alive(p)$ for every $p \in L_0$*
- *$sort(p,s)$ for every $p \in L_0$ and $s \in \{\bullet, \otimes, ⅋\}$,*
- *$edge(p,0,q,i)$ for every $p,q \in L_0$ and $i \leq |L_0|$.*

A link $p \in L_0$ is said to be alive *in θ if $alive(p) = 1$. Given a proof net $P = <L, \sigma, \sim>$, we write $\theta \in \mathsf{Conf}(P)$ if for every $p \in L_0$, $alive(p) = 1 \Longleftrightarrow p \in L$, and for every alive links p and q in θ, the following hold:*

$$sort(p,s) = 1 \iff \sigma(p) \text{ is of sort } s;$$
$$edge(p,0,q,i) = 1 \iff (p,0) \sim (q,i).$$

Here, we call a $\otimes^n$-link ($⅋^n$-link, resp.) *of sort* $\otimes$ ($⅋$, resp.), forgetting the arities of $\otimes$ and $⅋$. It is clear that the number of Boolean values in a configuration is $O(|L_0|)^3$.

We are now ready to simulate each reduction step.

Lemma 19 *There is an unbounded fan-in circuit C of size $O(|P_0|^3)$ and constant depth such that whenever a configuration $\theta \in \mathsf{Conf}(P)$ is given as input and $P \Longrightarrow_m P'$, C outputs a $\theta' \in \mathsf{Conf}(P')$. The same holds for $\Longrightarrow_a$ too.*

Lemma 20 *There is an unbounded fan-in circuit C with $st\text{CONN}_2$ gates which is of size $O(|P_0|^3)$ and constant depth such that whenever a configuration $\theta \in \mathsf{Conf}(P)$ is given as input and $P \Longrightarrow_t P'$, C outputs a $\theta' \in \mathsf{Conf}(P')$.*

Proof. Given $\theta \in \mathsf{Conf}(P)$, define

- $mid(p) = 1$ iff p is an alive axiom-link in θ and there are alive axiom-links q, r such that $(q,0) \sim (p,0) \sim (r,0)$
- $end(p) = 1$ iff p is an alive axiom-link in θ and $mid(p) = 0$
- $aseq(p,q) = 1$ iff $end(p) = end(q) = 1$ and there is an a-sequence of alive links from p to q.

The conditions on the right hand side can actually be spelled out in terms of Boolean values in θ.

Note in particular that $aseq(p,q)$ can be computed in constant depth by using a $st\text{CONN}_2$ gate (applied to the undirected graph which consists of the vertices L_0 and the edges between alive axiom-links in θ). Now one can compute a $\theta' \in \mathsf{Conf}(P')$ which consists of the following values:

$$\begin{aligned}
alive'(p) &= alive(p) \wedge \neg mid(p) \wedge \neg \bigvee_{q \prec p} aseq(q,p) \\
sort'(p,s) &= sort(p,s) \\
edge'(p,0,q,i) &= edge(p,0,q,i) \vee \\
&\quad \bigvee_{r \succ p} (aseq(p,r) \wedge edge(r,0,q,i)).
\end{aligned}$$

The computation above can be easily implemented by a constant depth circuit. ■

Theorem 21 *For every Boolean proof net P of size s and depth d, there is a boolean circuit C of size $O(s^4)$ and depth $O(d)$ over the basis $\mathscr{B}_1(st\text{CONN}_2)$ which accepts the same set as P does.*

Proof. Almost clear from Theorem 17, Lemma 19 and Lemma 20. It just remains to show that there is a constant depth circuit for initialization which computes a $\theta \in \mathsf{Conf}(P(\mathsf{b}_{i_1},\ldots,\mathsf{b}_{i_n}))$ from given inputs $i_1,\ldots,i_n$, and there is a constant depth circuit for acceptance checking which decides whether a given configuration θ' represents b_1 or not. These are easily constructed. ■

6. Proof Net Complexity

Analogously to the *AC* hierarchy, we define a hierarchy of complexity classes based on proof nets.

Definition 22 (*APN* **hierarchy**) *A family $\{P_n\}_{n\in\mathbb{N}}$ of Boolean proof nets* accepts *a language $X \subseteq \{0,1\}^*$ if P_n is n-ary and accepts $X \cap \{0,1\}^n$ for every $n \geq 1$.*

A language $X \subseteq \{0,1\}^*$ belongs to the class APN^i iff X is accepted by a polynomial size, $\log^i$-depth family of Boolean proof nets. More precisely, $X \in APN^i$ iff X is accepted by a family $\{P_n\}_{n\in\mathbb{N}}$ of Boolean proof nets and there is some k such that each P_n is of size $O(n^k)$ and of depth $O(\log^i n)$.

We finally end up with:

Theorem 23

1. *$APN^i = AC^i(st\text{CONN}_2)$ for every $i \geq 0$.*

2. *$\bigcup_{i\in\mathbb{N}} APN^i = NC$.*

3. *$P/poly$ = the class of those languages for which there is a polynomial size family of Boolean proof nets.*

7 Conclusion and Future Work

We have established a connection between proof nets and Boolean circuits by giving depth-preserving simulations of each other. The connection is based on the proofs-as-programs paradigm, in contrast to the well-known correspondences between circuit complexity and propositional proof systems such as Frege systems (see [9]). Our study has led to a precise characterization of parallel complexity of proof nets, namely $APN^i = AC^i(st\text{CONN}_2)$. To establish this result, we have employed a graph-theoretic reasoning (§4.1), and used some higher order functionals (§4.2). These techniques are further to be explored. Another research direction would be to study the bounded fan-in case, which seems to be more difficult. It should also be interesting to enrich proof nets with additives, because additives allow us to incorporate nondeterminism [13].

Finally, we hope that our study will shed a new light on the relationship between parallel-nonuniform computation (exemplified by Boolean circuits) and sequential-uniform computation (exemplified by functional programming/constructive logics, from which proof nets stem).

Acknowledgments
We would like to thank Noriko Arai, Harry Mairson and the anonymous referees for helpful suggestions. We are especially indebted to one of the referees who suggested an encoding of $st\text{CONN}_2$ in Section 4.2.

References

[1] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 757–804. Elsevier, 1990.

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

[3] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

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

[5] J.-Y. Girard. Proof-nets: The parallel syntax for proof-theory. In P. Agliano and A. Ursini, editors, *Logic and Algebra*. Marcel Dekker, New York, 1996.

[6] S. Guerrini. Correctness of multiplicative proof nets is linear. In *Proceedings of LICS 1999*, pages 454–463, 1999.

[7] D. J. D. Hughes and R. J. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic (extended abstract). In *Proceedings of LICS 2003*, pages 1 – 10, 2003.

[8] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *Journal of Discrete Mathematics*, 3:255–265, 1990.

[9] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, 1995.

[10] Y. Lafont. From proof nets to interaction nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995.

[11] O. Laurent. Polarized proof-nets and lambda-mu calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.

[12] H. Mairson. Linear lambda calculus and polynomial time. *Journal of Functional Programming*, to appear.

[13] H. Mairson and K. Terui. On the computational complexity of cut-elimination in linear logic. In *Proceedings of ICTCS 2003*, pages 23–36. LNCS 2841, 2003.

[14] A. A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki*, 41:598–607, 1987.

[15] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of 19th Symposium on Theory of Computing*, pages 77–82. ACM Press, 1987.

[16] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1998.

[17] I. Wegener. *The Complexity of Boolean Functions*. B. G. Teubner & John Wiley, 1987.