Uniform circuits, & Boolean proof nets

Virgile Mogbil[1]* and Vincent Rahli[2]*

[1] LIPN – UMR7030, Université Paris 13 – CNRS, France
[2] ULTRA, Heriot-Watt University, Scotland

Abstract. The relationship between Boolean proof nets of multiplicative linear logic (APN) and Boolean circuits has been studied [Ter04] in a non-uniform setting. We refine this results by taking care of uniformity: the relationship can be expressed in term of the (Turing) polynomial hierarchy. We give a proofs-as-programs correspondence between proof nets and deterministic as well as non-deterministic Boolean circuits with a uniform depth-preserving simulation of each other. The Boolean proof nets class $m\&BN(poly)$ is built on multiplicative and additive linear logic with a polynomial amount of additive connectives as the non-deterministic circuit class $NNC(poly)$ is with non-deterministic variables. We obtain uniform-$APN = NC$ and $m\&BN(poly) = NNC(poly) = NP$.

1 Introduction

Linear Logic (LL) is a refinement of classical and intuitionist logic [Gir87]. The conjunction/disjunction are split into the multiplicative (M) and additive (A) connectives. The exponentials give a logical status to the structural rules of classical and intuitionist sequent calculus. The study of LL revealed the *proof nets* [Gir87,DR89]: a parallel syntax for logical proofs where some inessential sequential information are removed. In this way the global and sequential sequent calculus cut-elimination becomes local and parallel in the proof nets. The well known Curry-Howard isomorphism is a correspondence between proofs and programs which associates cut-elimination in proofs and execution in programs. We study its extension to models of parallel computation using proof nets.

Boolean circuits [Vol99,BS90] are a standard model of parallel computation as Turing machine are a model of sequential computation. Several important complexity classes are defined in terms of Boolean circuits, including NC. NC can be thought of as the problems being efficiently solved on a parallel computer just as the class P can be thought of as the tractable problems. Because a circuit has a fixed input size, Boolean circuits are so-called non-uniform models of computation: inputs of different lengths are processed by different circuits. A *uniformity* condition is often imposed on circuit families so that each circuit can be computed by some resource-bounded Turing machine. For instance NC is defined to be the set of Boolean functions that can be decided by uniform Boolean circuits of polynomial size in the length of the inputs and polylogarithmic depth.

* Work partially supported by projects GEOCAL (ACI) and NO-CoST (ANR)

The depth is the time on a parallel computer where the size is the number of processors.

K. Terui (NII, Japan) introduced a proof-as-programs correspondence between a multiplicative Boolean proof nets class (APN) and Boolean circuits such that cut-elimination corresponds to evaluation [Ter04]. Defining a parallel cut-elimination in proof nets, the Terui's main result is APN = non-uniform NC. This is the first time that a logical depth is taken into account: it makes possible to achieve a seed up over sequential computation. Without restricting the depth in NC or non-uniform NC, one obtains respectively P or $P/poly$. $P/poly$ is the complexity class of languages recognized by a polynomial-time Turing machine with a polynomial-bounded advice function. So K. Terui gives a corollary: polynomially-sized multiplicative Boolean proof nets class is equivalent to $P/poly$. Our paper is firstly motivated by an algorithmic point of view, and therefore an important issue is the uniformity of circuits, because only a uniform circuit family $(C_n)_{n\in\mathbb{N}}$ can be regarded as an implementation of an algorithm. Indeed a description of the circuit C_n for inputs of size n can be obtained easily when the value of n is known: we need an efficient algorithm to built C_n given n, where different notions of efficient give rise to different notions of uniformity [Ruz81,All89,BIS90]. We introduce a suitable notion of uniformity for proof nets and adapt the proofs between proof nets and Boolean circuits to satisfy the uniformity condition. The method gives the uniform counter part of the Terui's results i.e. a proof nets characterisation of both NC and P.

Note that $P/poly$ is not generally considered a practical class for computing. Indeed it contains every undecidable unary language, none of which can be solved in general by real computers. However $P/poly$ is an important theoretical class because of the following fact: if $NP \subseteq P/poly$ then the polynomial hierarchy collapses to $\Sigma_2 P$ (i.e. NP with NP Oracle), and if NP is not a subset of $P/poly$ then $P \neq NP$ [KL80,CK06]. Our work is motivated by such theoretical point of view and the logical study of complexity classes. We study a non-deterministic extension of the parallel Curry-Howard isomorphism in a uniform setting always by giving a uniform depth-preserving simulation of each classes. On one hand there are several characterizations of non-determinism in circuits [Ven92,Wol94]. We use $NNC(poly)$ a class equivalent to NP, which is defined in the same way as NC but using at most a polynomial amount of non-deterministic variables. On the other hand, as suggested by K. Terui we enrich proof nets with additive connectives, because additive allow us to incorporate non-determinism. An encoding of a *co-NP* problem in the intuitionist fragment of MALL [MT03] illustrates why additives could be used. Contrary to the multiplicative case, the proof nets with additives have never been convincing: in the original syntax [Gir87], the additive connective & is associated to a box and the cut-elimination does not satisfy the Church-Rosser property. Nevertheless if a such proof net does not contain the connective & in its conclusions then it has a unique normal form [Tor03]. Our encoding and the defined Boolean proof nets satisfy this property, so we restrict our attention to this setting. Strong normalization and confluence of the additive proof nets have been studied in various directions [Tor03], in the

polarized fragment of LL [LdF04] or recently with a set of linkings on a formula [HvG03,HvG05]. We have not yet fully explored this last approach which seems to give another kind of speed-up.

After some background on Boolean circuits in section 2, we present the Terui's approach extended to the uniform case in section 3. The circuit gates are unbounded like $(\wedge^n)_{n\in\mathbb{N}}$ and $(\vee^n)_{n\in\mathbb{N}}$. In particular the $stCONN_2$ gates which test the reachability between two nodes of the undirected graph given in input, are used to simulate the cut-elimination of proof nets. In the first subsection the definitions concerning MLL_u (a n-ary variant of MLL) and Boolean type are presented as in [Ter04]. We define the uniformity for Boolean proof net families: the class mBN denotes just uniform-APN. Then, we improve the Terui's results with two theorems (Thm.2, Thm.5): the translation and simulation between NC and mBN are done in logspace. Section 4 is devoted to the uniform non-deterministic Boolean proof nets called $m\&BN(poly)$. We define the proof nets for $\mathrm{M}_u\mathrm{ALL}$ which is the fragment MLL_u extended with additive connectives. We remind the standard definitions of the additives connectives: the slices and the cut-elimination. We easily obtain a parallel reduction theorem because of the particular setting. We introduce an extended version of the Boolean type for the non-deterministic variables. In the last section we establish the translation and the simulation theorems which imply $NNC(poly) = m\&BN(poly) = NP$.

2 Background

– Boolean circuits –

Let F_n denote the set of all Boolean functions $f : \{0,1\}^n \to \{0,1\}$ for some $n \in \mathbb{N}$. A *basis* is a finite set consisting of Boolean functions or sequences of Boolean functions $(f_i)_{i\in\mathbb{N}}$ where $f_i \in F_i$. The standard basis are $\mathcal{B}_0 = \{\neg, \wedge, \vee\}$ and $\mathcal{B}_1 = \{\neg, (\wedge^n)_{n\in\mathbb{N}}, (\vee^n)_{n\in\mathbb{N}}\}$.

A *Boolean circuit* over a basis $\mathcal{B}$ is a directed acyclic graph with $n+1$ sources or inputs (vertices with no in-going edges), one sink or output (a vertex with no out-going edges) and all nodes in $\mathcal{B}$. Sources are labelled by literals from $\{x_1, \ldots, x_n\} \cup \{1\}$ and nodes of in-degree k are labelled by one of the k-ary Boolean functions of $\mathcal{B}$. A Boolean circuit (a circuit for short) computes a function in F_n in a natural way. Nodes are called *gates*, and in-degree and out-degree are called *fan-in* and *fan-out* respectively. The circuits over basis without infinite families of Boolean functions (as $\mathcal{B}_0$), are called *bounded fan-in* circuits. The other circuits are called *unbounded fan-in* circuits.

We say that a *family of circuits* $C = (C_n)_{n\in\mathbb{N}}$ computes a function $f : \{0,1\}^* \to \{0,1\}$ (or recognizes a language $L_C \in \{0,1\}^*$) if for every n the circuit C_n computes the restriction of f to F_n. I.e. $\forall x \in \{0,1\}^*, C_{|x|}(x) = f(x)$.

Let C be a circuit, the *size* denoted $size(C)$ is the number of gates of C. The *depth* denoted $d(C)$ is the length of a longest directed path.

A *non-deterministic* Boolean circuit C with m non-deterministic variables is a circuit with $n+m+1$ sources labelled by $\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\} \cup \{1\}$.

It computes a function $f \in F_n$ as follows: for $x \in \{0,1\}^n$, $f(x) = 1$ iff there exist a setting of the non-deterministic variables $\{y_1, \ldots, y_m\}$ which makes the circuit output 1. We denote $C(x, y)$ the same circuit as C without distinction between non-deterministic variables and deterministic gates, $x \in L_C$ the language recognized by C, if $\exists w \in \{0,1\}^m$ a witness s.t. $C(x, w) = 1$. When needed we abusively denote $C(x)$ and $C(x, y)$ the distinct circuits.

– Circuit uniformity –

As briefly presented in the introduction, there are different notions of circuit uniformity [Vol99]. In order to obtain a class containing P, it is necessary to impose a "P-uniform" condition on circuit families. That is, the description of the n^{th} circuit can be provided by a deterministic Turing machine operating in polynomial time [All89]. But P-uniformity is too weak to define subclasses of P; this leads one to consider L-uniformity [Ruz81]: the description is computable in logspace. It is the same when we want to consider the subclass of NC with $O(log\ n)$ depth (called NC^1): *DLOGTIME*-uniformity requires a somewhat more careful definition. Informally, there is a deterministic linear time in $O(log\ s(C_n))$ Turing machine that, given n and a name of a gate g can determine all the wanted information about gate g (like sort, predecessors, ...) belonging to the circuit C_n of size s. Unfortunately all these notions are more and more restrictive.

In our work we focus on a uniform approach of the well established relationship between non-uniform NC and the multiplicative Boolean proof net class APN. For the sake of simplicity we consider the L-uniformity: it is sufficient to investigate all classes containing L. Actually, only NC^1 and constant depth classes of circuits and proof nets need a uniformity notion stronger than the L-uniformity. Moreover the NC class remains the same if stronger notions of uniformity than L-uniformity are used [Ruz81].

The *direct connection language* of a family $C = (C_n)_{n\in\mathbb{N}}$ over basis $\mathcal{B}$, denoted $L_{DC}(C)$, is the set of tuples $\langle y, g, p, b\rangle$, where for $y = 1^n$, we have: g is the number of a gate v in C_n, and $p \in \{0,1\}^*$ is a binary word such that
- if $p = \varepsilon$ then b is the number of the function from $\mathcal{B}$ labeling v,
- if $p = bin(k)$ then b is the number of the k^{th} predecessor gate to v.

A circuit family $(C_n)_{n\in\mathbb{N}}$ is *L-uniform* if its direct connection language can be recognized in logspace by a deterministic Turing machine. Without precisions, we use in the rest of this paper the term *uniform* as a shorthand for L-uniform.

– Circuit classes –

The *classes* NC^i and AC^i for $i \geqslant 0$ are the functions computable by uniform families of polynomial size, $O(log^i n)$ depth circuits over $\mathcal{B}_0$ and $\mathcal{B}_1$ respectively. $AC^i(stCONN_2)$ correspond to AC^i over $\mathcal{B}_1 \cup \{stCONN_2\}$. We denote NC the uniform circuit families which have polynomial size and polylogarithmic depth, i.e. $NC = \cup_{i\geqslant 0} NC^i$. We define AC in the same way. L, NL and P are in the time-space hierarchy of the Turing machines. The well known hierarchy is:

$$AC^0 \subsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1 \subseteq NC^2 \subseteq AC^2 \subseteq \cdots \subseteq NC \subseteq P$$
$$\forall i \in \mathbb{N}, AC^i \subseteq AC^i(stCONN_2) \subseteq AC^{i+1}$$

The *class* $NNC^i(f(n))$ is the class of languages accepted by L-uniform-NC^i circuit families with at most $O(f(n))$ non-deterministic variables, where n is the length of the input. We define $NAC^i(f(n))$ in the same way, but using AC^i. We abusively denote $NNC^i(poly)$ when $f(n)$ is a polynomial function. If $f(n) = log\, n$ then the amount of non-deterministic variables can be described by a polynomial number of NC gates [Wol94]:

$$NNC^i(log\ n) = NC^i, \text{ and then } NNC(log\ n) = \cup_{i\geqslant 0} NNC^i(log\ n) = NC.$$

So we don't consider these classes but we investigate the following classes [Wol94]:

$$NNC(poly) = \cup_{j\geqslant 0} NNC(n^j) = NP \text{ and } \forall i \in \mathbb{N}^*, NNC^i(poly) = NP.$$
$$NAC(poly) = \cup_{j\geqslant 0} NAC(n^j) = NP \text{ and } \forall i \in \mathbb{N}, NAC^i(poly) = NP.$$

3 Uniform Boolean proof nets

– MLL_u and Boolean type –

We recall in this section some basic definitions gave by Terui in [Ter04]. The *formulas* of MLL_u are built on literals by n-ary versions of multiplicative conjunction and disjunction, for every $n \geqslant 2$. The negation of a non-literal formula is defined by de Morgan's duality (reversing the order of subformulas).

A *sequent* of MLL_u is of the form $\vdash \Gamma$, where Γ is a multiset of formulas. The rules of MLL_u are given in Fig.1(a) with the convention $\overrightarrow{A} \equiv A_1, \ldots, A_n$ and $\overleftarrow{A} \equiv A_n, \ldots, A_1$.

(a)
$$\frac{}{\vdash A, A^\perp}\ (axiom) \qquad \frac{\vdash \Gamma, C \quad \vdash \Delta, C^\perp}{\vdash \Gamma, \Delta}\ (cut)$$
$$\frac{\vdash \Gamma_1, A_1 \quad \ldots \quad \vdash \Gamma_n, A_n}{\vdash \Gamma_1, \ldots, \Gamma_n, \otimes^n(\overrightarrow{A})}\ \otimes^n \qquad \frac{\vdash \Gamma, A_n, \ldots, A_1}{\vdash \Gamma, \parr^n(\overleftarrow{A})}\ \parr^n$$

(b)
$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A\&B}\ \& \qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus A_2}\ \oplus_{i=1,2}$$

Fig. 1. (a) MLL_u (a+b) $\mathrm{M}_u\mathrm{ALL}$

The corresponding *links* (Fig.2(a)) are of three *sorts* called: axiom-link, $\otimes^n$-link and $\parr^n$-link. Each link has several ports. The ports numbered 0 are called the principal ports, while others are called auxiliary ports. By convention a principal port is always written below the link whereas an auxiliary port is above it.

A *pseudo net* is a triple $\langle L, \sigma, \sim\rangle$ s.t. L is a finite set of links, $\sigma : L \to \{\bullet\} \cup \{\otimes^n, \parr^n\}_{n\geqslant 1}$ and $\sim$ is a symmetric relation on $(L, \mathbb{N})^2$.

A link p with $\sigma(p) = \bullet$ ($\otimes^n$, $\parr^n$, resp.) stands for an axiom-link ($\otimes^n$-link, $\parr^n$-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 the port number n of link p. A *cut* in a pseudo

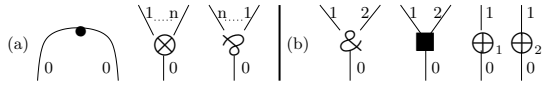


Fig. 2. (a) Multiplicative and (b) additive links

net is an unordered pair of link p, q s.t. $(p, 0) \sim (q, 0)$: we call it an ax-cut when either p or q is an axiom-link, otherwise we call it a m-cut.

A *proof net* of type $\vdash \Gamma$ is a pseudo net P inferred by a sequent calculus proof of $\vdash \Gamma'$ where Γ is a decoration of Γ' i.e. formulas of the form $p : A$ (see Fig. 5 for an example) (a proof net of type $\vdash p : A$ is simply called a proof net of type A) . The pseudo nets inferred in Fig.3(a) are respectively: $tensor_q^{p_1,\ldots,p_n}(P_1, \ldots, P_n)$, $par_q^{p_n,\ldots,p_1}(P)$ and ax_p, $cut^{p,q}(P, Q)$.

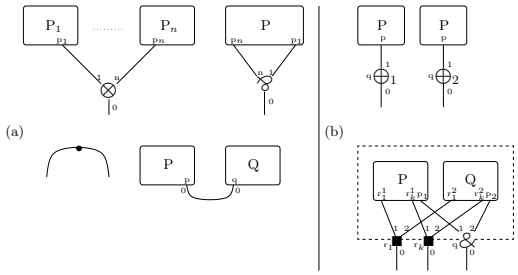


Fig. 3. (a) Multiplicative and (b) additive proof net constructors

The *size* $|P|$ is the number of links in P. The *depth* $d(A)$ of a formula A is given by $d(\alpha) = d(\alpha^\perp) = 1$ and $d(\otimes^n(\overrightarrow{A})) = d(\parr^n(\overleftarrow{A})) = max(d(A_1), \ldots, d(A_n)) + 1$, with $\overrightarrow{A} \equiv A_1, \ldots, A_n$. Given a derivation π of $\vdash \Gamma$ inferring P, its depth $d(\pi)$ is the maximal depth of cut formulas in it. The *depth* $d(P)$ of a proof net P is defined to be $min\{d(\pi) | \pi$ is a derivation of $\vdash \Gamma$ inferring P for some $\Gamma\}$.

Boolean values are represented with the type $\mathbf{B} = \parr^3(\alpha^\perp, \alpha^\perp, \otimes^2(\alpha, \alpha))$. There are exactly two cut-free proof nets of this type (*true* and *false* resp.) Fig.4(a): $b_1 \equiv par_s^{p,q,r}(tensor_r^{p,q}(ax_p, ax_q))$, $b_0 \equiv par_s^{q,p,r}(tensor_r^{p,q}(ax_p, ax_q))$.

A *Boolean proof net* with n inputs $\overrightarrow{p} \equiv p_1, \ldots, p_n$ and one output is a proof net $P(\overrightarrow{p})$ of type: $\vdash p_1 : \mathbf{B}^\perp[A_1], \ldots, p_n : \mathbf{B}^\perp[A_n], q : \otimes^{m+1}(\mathbf{B}, \overrightarrow{C})$, for some $\overrightarrow{A} \equiv A_1, \ldots, A_n$ and $\overrightarrow{C} \equiv C_1, \ldots, C_m$ (garbage due to the multiplicative framework) where $\mathbf{B}[A]$ denote the formula $\mathbf{B}$ where all occurrences of α are substituted by A. Given $\overrightarrow{b} \equiv b_{i_1}, \ldots, b_{i_n}$, $P(\overrightarrow{b})$, of type $\otimes^{m+1}(\mathbf{B}, \overrightarrow{C})$, denotes the proof net obtained by connecting, $\forall j \in \{1, \ldots, n\}$ ($i_j \in \{0, 1\}$), b_{i_j} to p_j by a cut. $P(\overrightarrow{b})$ reduces to a cut-free proof net of the shape $tensor(b_i, \overrightarrow{Q})$, $i \in \{0, 1\}$: we say that $P(\overrightarrow{b})$ evaluates to b_i. Let $w \equiv i_1 \ldots i_n \in \{0, 1\}^n$. $P(\overrightarrow{p})$ represents a function $f : \{0, 1\}^n \to \{0, 1\}$ if $P(b_{i_1}, \ldots, b_{i_n})$ evaluates to $b_{f(w)}$. Thus, the language accepted by $P(\overrightarrow{p})$ is $f^{-1}(1)$.

APN^i for $i \in \mathbb{N}$, is the class of languages recognized by non-uniform Boolean proof net families of polynomial size and $O(log^i\ n)$-depth. APN= $\bigcup_{i\in\mathbb{N}} APN^i$.

– Uniform proof nets –

In the framework of polynomial size proof nets, we consider an extended description of a proof net P which is equivalent to the triple defining a pseudo net. We call it $Conf(P)$, the configuration of P.

Let $P = \{P_n\}_{n\in\mathbb{N}}$ be a family of Boolean proof nets. Links are identified by binary words. For all $n \in \mathbb{N}$, we fix the inputs $p_1, \ldots, p_n$ of P_n to be identified by $0, \ldots, bin(n-1)$ respectively, and the output to be identified by $bin(n)$ (where bin is the function which associates to a number in decimal base its value in binary base). *Conf(P)* denotes the set of tuples in $\{1\}^* \times (W \setminus \{\epsilon\}) \times W^2 \times \{0,1\}$ (W is the set of binary words), s.t. for $y = 1^n$:

- in $\langle y, u, \epsilon, \epsilon, 1\rangle$, u identifies a P_n link.
- in $\langle y, u, s, \epsilon, 1\rangle$, s is the sort's identifier of the link identified by u.
- in $\langle y, u, v, bin(i), 1\rangle$, u identifies a link connected by its principal port (or one of its two principal ports in the case of an axiom) to the port i of the link identified by v.
- in $\langle y, u, a, b, 0\rangle$, u, a, b are the same informations as above but are not concerned with P_n (i.e. the edges which does not appear in P_n).

Remark that if $P_n \in P$ then $Conf(P_n)$ is the set of tuples that belong to $Conf(P)$ s.t. the first word is of size n. A proof net family $\{P_n\}_{n\in\mathbb{N}}$ of polynomial size s is *L-uniform* (resp. *P-uniform*) iff there is a function which computes $Conf(P_n)$ from 1^n in space $O(log\ s)$ (resp. in time $s^{O(1)}$), for all $n \in \mathbb{N}$. For all $i \in \mathbb{N}$, uniform APN^i is denoted mBN^i.

– Uniform Terui's translation of NC –

The *conditional* (if-then-else, Fig.4(b)) is the base of the Terui's gates translations: given two proof nets P_1 and P_2 of types $\vdash \Gamma, p_1 : A$ and $\vdash \Delta, p_2 : A$ resp., one can build a proof net $cond_r^{p_1,p_2}[P_1, P_2](q)$ of type $\vdash \Gamma, \Delta, q : \mathbf{B}[A]^\perp, r : A \otimes A$. Given a cut between b_i and q we have with the convention that the first component is considered as the output, the rest being the garbage:

$$cond_r^{p_1,p_2}[P_1, P_2](b_1) \to^* tensor_r^{p_1,p_2}(P_1, P_2),$$
$$cond_r^{p_1,p_2}[P_1, P_2](b_0) \to^* tensor_r^{p_2,p_1}(P_2, P_1).$$

Disjunction, *conjunction* and *duplication* are based on the conditional: let $n \geqslant 2$ be an integer and $C \equiv \otimes^n(\mathbf{B}[A_1], \ldots, \mathbf{B}[A_n])$,

$$or(p_1, p_2) \equiv cond[b_1, ax_{p_1}](p_2) \text{ of type } \vdash p_1 : \mathbf{B}^\perp, p_2 : \mathbf{B}[\mathbf{B}]^\perp, q : \mathbf{B} \otimes \mathbf{B},$$
$$and(p_1, p_2) \equiv cond[ax_{p_1}, b_0](p_2) \text{ of type } \vdash p_1 : \mathbf{B}^\perp, p_2 : \mathbf{B}[\mathbf{B}]^\perp, q : \mathbf{B} \otimes \mathbf{B},$$
$$copy^n(p) \equiv cond[tensor(\overrightarrow{b_1}), tensor(\overrightarrow{b_0})](p) \text{ of type } \vdash p : \mathbf{B}^\perp[C], q : C \otimes C.$$

The *composition* (Fig.4(c)) of two translated circuits is defined as follows: let $\Gamma \equiv p'_1 : A'_1, \ldots, p'_n : A'_n$ and $\Delta \equiv q'_1 : B'_1, \ldots, q'_n : B'_m$, let $P(\overrightarrow{p'})$ and $Q(\overrightarrow{q'})$ be

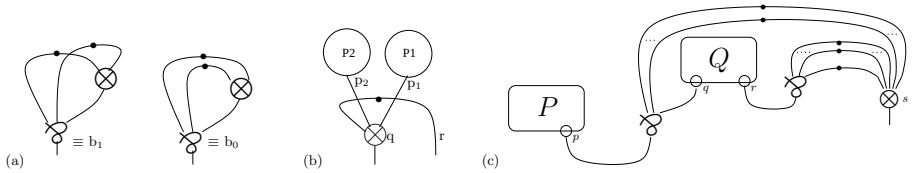


Fig. 4. (a) The Boolean b_1 and b_0 (b) The conditional (c) The composition

proof nets of type $\vdash \Gamma, p : \otimes^{1+m}(\mathbf{B}, \overrightarrow{C})$ and $\vdash q : \mathbf{B}^{\perp}[A], \Delta, r : \otimes^{1+m'}(\mathbf{B}, \overrightarrow{D})$, respectively. Then:
$comp_s^{p,q,r}[P,Q](\overrightarrow{p'}, \overrightarrow{q'})$ is of type $\vdash \Gamma[A], \Delta, s : \otimes^{1+m'+m}(\mathbf{B}, \overrightarrow{D}, \overrightarrow{C[A]})$.

With this composition one can construct n-ary versions of conjunction and disjunction. The translation of a Boolean circuit follows from composition of gate translations and duplication for fan-out management:

Theorem 1 ([Ter04]). *For every unbounded fan-in Boolean circuit C of size s and depth d over the basis $\mathscr{B}_1(stCONN_2)$, there is a Boolean proof net of size $O(s^5)$ and depth $O(d)$, which accepts the same set as C does.*

Lemma 1. *Let Σ be a finite alphabet. If $g : \Sigma^* \to \Sigma^*$ and $f : \Sigma^* \to \Sigma^*$ are computable in logspace and the f output is polynomial in its input, then $g \circ f$ is computable in logspace.*

Theorem 2. *Let $i \in \mathbb{N}$. The Terui translation of a Boolean circuit family in $AC^i(stCONN_2)$, by a Boolean proof net family in mBN^i is logspace.*

Remark 1. In order to simplify this result we consider a translation of a circuit where the garbage of each translated gate is propagated directly to the output of the proof net. This version is equivalent to the one presented by Terui but in which some reductions have been performed.

Proof. Let $C = \{C_n\}_{n\in\mathbb{N}}$ be a L-uniform Boolean circuit family in $AC^i(stCONN_2)$. By uniformity, there is a logspace function f s.t. for every $n \in \mathbb{N}$, $f(1^n) = L_{DC}(C_n)$. Let $P = \{P_n\}_{n\in\mathbb{N}}$ be the Boolean proof net family obtained by translation of C. We show that there is a logspace function f' built from f s.t. for every $n \in \mathbb{N}$, $f'(1^n) = Conf(P_n)$. In order to do that, we use a function $f_{d\to c}$, logspace in the inputs of f, which associates $Conf(P_n)$ to $L_{DC}(C_n)$ for all $n \in \mathbb{N}$. Let $f_{d\to c} = f_3 \circ f_2 \circ f_1$ where we call module the composition of the translated gate and its translated fan-out, and $(k, j \in \{1, \ldots, size(C_n)\})$:

- f_1 copies out its inputs adding to tuples which indicate that a gate v is the k^{th} predecessor of a gate u, that gate u is the j^{th} successor of gate v.
  Then f_1 computes for every gate its module composed with a pseudo net allowing garbage propagation, adding the identifier of the translated gate, and for each output, an identifier differentiating it from others.
- f_2 copies out and completes the information given by the modules created by f_1 with the help of the information added to $L_{DC}(C_n)$. That is, it adds

information about edges between an output of a module and an input of another module, with the help of information added about translated gates and edges between gates.
- f_3 organizes tuples and deletes useless information.

$L_{DC}(C_n)$ like $Conf(P_n)$ has a polynomial size. An identifier of a gate or a link is logspace. At each step, these functions memorize a constant number of identifiers, so f_1, f_2 and f_3 are logspace. Then $f_{d\to c}$ is logspace. By lemma 1, $f' = f_{d\to c} \circ f$ is logspace in the inputs of f. □

– Parallel cut-elimination –

The elimination of a cut between two axioms cannot always be performed in parallel. So K. Terui considers another reduction step named *tightening reduction* which reduces a maximal sequence of cut axioms to an axiom. Such maximal sequence (ax-sequence) is defined as a set of axioms, each linked with another in this set s.t. there are not other axioms that verify this property.

If Q is obtained from P by elimination of all ax-cuts simultaneously (m-cuts, ax-sequences, resp.) then we write $P \Rightarrow_{ax} Q$ ($P \Rightarrow_m Q$, $P \Rightarrow_t Q$, resp.). We write $P \Rightarrow Q$ if $P \Rightarrow_{ax} Q$ or $P \Rightarrow_m Q$ or $P \Rightarrow_t Q$.

Theorem 3 (parallel cut-elimination, [Ter04]). *There is a sequence of parallel reductions $P \Rightarrow P_1 \cdots \Rightarrow P_k$, s.t. P_k is cut-free and $k \leqslant 3 \times d(P)$.*

– Uniform Terui's simulation in NC –

We consider proof nets with links belonging to a fixed set L_0 with the convention $\parr^n$-link is of sort $\parr$, and a $\otimes^n$-link is of sort $\otimes$. A *configuration* Θ consists of the following Boolean values : $alive(p)$, $sort(p, s)$, $edge(p, 0, q, i)$, for every $p, q \in L_0$, $s \in \{\bullet, \otimes, \parr\}$ and $i < |L_0|$.

A link $p \in L_0$ is said to be alive in Θ if $alive(p) = 1$. Given a proof net $P = \langle L, \sigma, \sim\rangle$, we write $\Theta \in Conf(P)$ if for every $p \in L_0$, $alive(p) = 1 \iff p \in L$ and for every alive links p, q in Θ, the following holds : $sort(p, s) = 1 \iff \sigma(p)$ is of sort s and $edge(p, 0, q, i) = 1 \iff (p, 0) \sim (q, i)$.

Lemma 2 ([Ter04]). *There is an unbounded fan-in Boolean circuit C of size $O(|P_0|^3)$ and constant depth with $stCONN_2$ gates s.t. whenever $\Theta \in Conf(P)$ is given as input of C and $P \Rightarrow P'$, it outputs a configuration $\Theta' \in Conf(P')$.*

Theorem 4 ([Ter04]). *For every Boolean proof net P of size s and depth d, there is a Boolean circuit C over the basis $\mathscr{B}_1(stCONN_2)$ of size $O(s^4)$ and depth $O(d)$ which accepts the same set as P does.*

Theorem 5. *Let $i \in \mathbb{N}$. The Terui simulation of a Boolean proof net family in mBN^i by a Boolean circuit family in $AC^i(stCONN_2)$, is logspace.*

Proof. Let $P = \{P_n\}_{n\in\mathbb{N}}$ be a L-uniform Boolean proof net family in APN^i. By uniformity there is a logspace function f s.t. for all $n \in \mathbb{N}$, $f(1^n) = Conf(P_n)$. Let $C = \{C_n\}_{n\in\mathbb{N}}$ be the Boolean circuit family obtained by simulation of P. We show that there is a logspace function f' built from f s.t. for every $n \in \mathbb{N}$,

$f'(1^n) = L_{DC}(C_n)$. In order to do that, we use a function $f_{c\to d}$, logspace in the inputs of f, which associates $L_{DC}(C_n)$ to $Conf(P_n)$ for all $n \in \mathbb{N}$. The simulation introduced by Terui is divided in three steps: creation of initial configuration, simulation of parallel cuts elimination and check of the the last configuration.

- $f_{c\to d}$ builds the part of the first configuration which represents the inputs of the proof net using the inputs of the Boolean circuit. It builds the part which represents the proof net using its input. Only a constant number of identifiers of links are memorized.
- $f_{c\to d}$ builds the circuit which simulates parallel cuts elimination. This part is only dependent of the size and depth of the proof net and not of its structure.
- $f_{c\to d}$ builds the circuit which check the result contained in the last configuration. This building is only dependent of the size of the proof net.

$Conf(P_n)$ like $L_{DC}(C_n)$ has a polynomial size. An identifier of a gate or a link is logspace. At each step, $f_{c\to d}$ memorizes a constant number of identifiers, so it is logspace. By lemma 1, $f' = f_{c\to d} \circ f$ is logspace in the inputs of f. □

Corollary 1. *$mBN = NC$ and P is the class of those languages for which there is a uniform polynomial size family of (multiplicative) Boolean proof nets.*

4 Non-deterministic Boolean proof nets $m\&BN()$

In this section we introduce the unbounded fan-in version of multiplicative linear logic (MLL_u) extended with the binary additive connectives. We denote it $\mathrm{M}_u\mathrm{ALL}$. The *formulas* of $\mathrm{M}_u\mathrm{ALL}$ are built from literals by MLL_u connectives and by binary additive conjunction $\&$ and additive disjunction $\oplus$. The *sequent calculus rules* are given in Fig.1(a+b).

The added *links* are called $\&$-link, $\oplus_1$-link and $\oplus_2$-link (Fig.2(b)). The *inference rules* for the new links follow the sequent calculus rules as expected. The resulting proof nets are depicted in Fig. 3(b). In a derivation, the two premises of a $\&$-rule, with the same context Γ, infer two proof nets called *components*. Each identical conclusion coming from the two components are merged with a binary co-additive-links (denoted ■-links or coad-links). So a $\&$-link arrives with an *additive box* bounding a (possibly empty) set of coad-links and the two components. This is described in Fig.3(b) with the associated ports. As previously the cuts are just edges between principal ports.

– Slices and additive cut-elimination –

A *slice* of a proof net P is a proof net $sl(P)$ which may contain some unary $\&_1$ and $\&_2$ links: for every $\&$-link one of the premises is chosen. Then the non-corresponding component, the additive box and the coad-links are erased. In a slice, the additive cut-elimination between a $\oplus_i$-link and a $\&_j$-link, for $i, j \in \{1, 2\}$, is simply to check the *consistency*: if $i = j$ then the additive links are removed propagating the cut on the unique premises.

We define as usual [Gir87] the *additive cut-elimination*, denoted $\rightarrow_a$. The cut-elimination between a $\&$-link and a $\oplus_i$-link ($i \in \{1,2\}$) is the choice of the i^{th} $\&$-link's premise and the same erasing as a slice. A cut-elimination with a coad-link is more complicated: the cut proof net is swallowed and duplicated in the additive box. The duplicated conclusions are merged with coad-links.

The cut-elimination with a coad-link is not confluent. A standard example is the proof net of type $\vdash A\&A, A^\perp$, cut to the proof net of type $\vdash A^\perp\&A^\perp, A$. This produces two possible proof nets of type $\vdash A\&A, A^\perp\&A^\perp$ where the $\&$-link of the additive box is one of the formulas, the other being a coad-link. Nevertheless the proof nets are stable by cut-elimination [Dan90,Tor03]. Roughly speaking if the conclusions of the proof nets are $\&$-free then the cut-elimination is confluent.

– Parallel reduction theorem –

It is difficult to have an additive cut-elimination from a parallel point of view (denoted $\Rightarrow_a$). We need a strategy e.g. as the parallel tightening reduction, one could want to reduce first the $\&$-links in parallel. But it is difficult to bound the number of parallel reduction steps because of the coad-links (it depends on maximal nesting of the additive boxes it contains). In the non-deterministic Boolean proof nets framework, it is somewhat different as explained after in lemma 3: proof nets are always cut against an additive witness s.t. this cut is eliminated equivalently to a slice choice. So we start the parallel reductions by choosing a slice, then additive cut-elimination becomes trivial and confluent: $\Rightarrow_a$ is only the disjoint consistency check done in parallel. We write $\Rightarrow$ if one of the parallel reduction occurs (i.e. $\Rightarrow_t, \Rightarrow_{ax}, \Rightarrow_m$ or $\Rightarrow_a$). Because in a slice each sequence of all distinct parallel reductions strictly decrease the depth, we have:

Theorem 6 (parallel cut-elimination). *For every consistent slice sl, there is a sequence of parallel reductions $sl(P) \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k$ s.t. P_k is cut-free and $k \leqslant 4 \times d(P) + 1$.*

– Extended Boolean type for non-deterministic variables –

We define a non-deterministic Boolean type as $\mathbf{B_e} = (\mathbf{B} \otimes \alpha^\perp), (\alpha\&\alpha)$. There

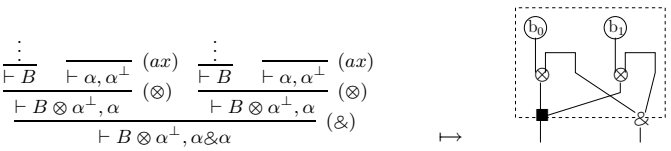


Fig. 5. From a sequent calculus proof of $\mathbf{B_e}$ to the chosen proof net

are four proof nets of this (yet strange) type. Two of them have the same behavior as the (deterministic) Boolean type $\mathbf{B}$. The two others can be arbitrarily used for our purpose: let the proof net depicted in Fig.5 be our choice of one of them. With this proof net we are able to internally choose between b_0 or b_1 using a slice

or equivalently by cut elimination with a witness as explained in the beginning of this section. Without loss of generality, for an arbitrary C, we allow an input type to be of the form $\mathbf{B_e}[C] = \mathbf{B_e}[\mathbf{B}[C]/\mathbf{B}, id/\alpha] = (\mathbf{B}[C] \otimes id^{\perp}) \parr (id \& id)$ where $id = \alpha \parr \alpha^{\perp}$ is a technical trick to simplify reductions in the translation and in the simulation.

– $m\&BN()$ description and hierarchy –

A *non-deterministic Boolean proof net* with n inputs and $O(f(n))$ additive links is a proof net $P(\overrightarrow{b})$ of type (for some $\overrightarrow{D} = D_1, \ldots, D_k$ and $m = f(n)$):

$$\vdash p_1 : \mathbf{B}^{\perp}[A_1], \ldots, p_n : \mathbf{B}^{\perp}[A_n], q : \otimes^{1+k+m}(\mathbf{B}, \overrightarrow{D}, \overrightarrow{id}), r : \parr^{m}(\overrightarrow{id^{\perp} \& id^{\perp}}).$$

Clearly, for w a proof net of type $\otimes^m(\overrightarrow{id \oplus id})$ and $\overrightarrow{b} \equiv b_{i_1}, \ldots, b_{i_n}$ $(\forall j \in \{1, \ldots, n\},\ i_j \in \{0,1\})$, the proof net $cut(P(\overrightarrow{b}), w)$ reduces to a cut-free proof net of type $\otimes^{1+k+m}(b_i, \overrightarrow{D}, \overrightarrow{id})$ $(i \in \{0,1\})$: we say that $P(\overrightarrow{b})$ evaluates to b_i if there exists such a w. The language accepted by a non-deterministic Boolean proof net is defined in the same way as for Boolean proof nets in section 3.

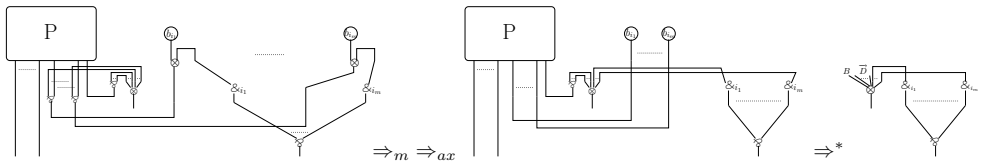
Similarly to the Terui APN hierarchy and to the $NNC()$ hierarchy, we define the $m\&BN()$ hierarchy: a uniform family $(P_n)_{n \in \mathbb{N}}$ of non-deterministic Boolean proof nets *accepts* a language $X \subseteq \{0,1\}^*$ if P_n is n-ary and accepts $X \cap \{0,1\}^n$ for every $n \geqslant 1$. A language $X \subseteq \{0,1\}^*$ belongs to the class $m\&BN^i(poly)$ iff X is accepted by a uniform polynomial size, log^i-depth family of non-deterministic Boolean proof nets with a polynomial number of additive links.

5 $NNC(poly) = m\&BN(poly)$

In order to obtain the equality between $NNC(poly)$ and $m\&BN(poly)$, we use the class $NAC(poly)$ whose relations with $NNC(poly)$ were described in section 2.

– translation of $NAC(poly)$ –

Let $C(x)$ be a circuit of input length n belonging to a family in $NAC^i(poly)$ for an arbitrary $i \in \mathbb{N}$. By definition $C(x,y)$ and $C(x)$ have the same dimensions and $m = \mid y \mid = n^{O(1)}$. Thus $C(x,y) \in AC^i$ w.r.t. the size of x, if we just omit the distinction between non-deterministic variables and deterministic variables. As in [Ter04] (see section 3) let P be the translation of $C(x,y)$. For each y we cut the built proof net P with exactly one representation of $\mathbf{B}_e$. We manage the garbage as it can be seen in Fig.6 to obtain the proof net T. Such *translated* proof net is in $m\&BN^i(poly)$. Let sl be a slice of T. We have the following parallel reduction from $sl(T)$:



Another way could be to choose a witness w, i.e a proof net of type $\otimes^m(\overrightarrow{id \oplus id})$, and then to reduce the cut between T and w. By only one additive parallel reduc-

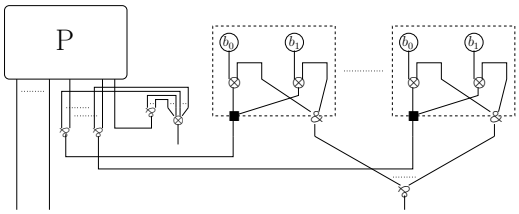


Fig. 6. translation

tion we obtain confluently a multiplicative proof net: it corresponds to $C(-, w)$. From the translation it immediately holds that to have a slice or a witness is equivalent in this setting, as is stated in this lemma:

Lemma 3. *Let $C \in NAC(poly)$ be a circuit family and let $P = (P_n)_{n\in\mathbb{N}}$ be the uniform non-deterministic Boolean proof net family obtained by the translation of C. Let $C(x, y) \in AC$ be the deterministic circuit family corresponding to C s.t. $|y| = m$. We have $x \in L_C$, the language recognized by C*
$\iff \exists w \in \{0,1\}^m$ *a witness s.t.* $C_n(x, w) = 1$
$\iff \exists sl$ *a slice of* P_n *s.t.* $sl(P_n(x)) \Rightarrow^* \otimes^{1+n+m}(b_1, \overrightarrow{D}, \overrightarrow{\alpha}), \parr_{1\leqslant j\leqslant m}(\&_{i_j}(\alpha))$
$\iff \exists w$ *a witness of type* $\otimes^m(\overrightarrow{id \oplus id})$ *s.t.* $cut(P_n(x), w) \Rightarrow^* \otimes^{1+l+m}(b_1, \overrightarrow{D}, \overrightarrow{id})$
$\iff x \in L_P$, *which is the language recognized by* P

Theorem 7 (translation). *Let $i \in \mathbb{N}$. For every Boolean circuit $C \in NAC^i(poly)$ of size s and depth d, there is a Boolean proof net in $m\&BN^i(poly)$ of size $O(s^5)$ and depth $O(d)$ which accepts the same set/language as C does.*

Proof. Let $C \in NAC^i(poly)$ be a circuit of input length n. Let $s = size(C)$ and $d = d(C)$. Let $m = |y| = n^{O(1)} \leqslant s$ be the amount of non-deterministic variables. Following Terui's theorem, every gate of fan-in f and fan-out o can be encoded by a proof net of size $O(f^4 + o) \leqslant O(s^4)$ and of constant depth. The depth increase is linear in d. Because $C(x, y)$ and C have the same dimensions, we obtain the same result as Terui. The last construction due to the non-deterministic variables translation doesn't change the proof net dimensions: we add $O(m + s)$ links and a constant to the depth. □

– Cut-elimination simulation –

Let $P \in m\&BN^i(poly)$ be a proof net of size s and depth d. Let $NAC^i(poly)$ $(stCONN)$ corresponds to $NAC^i(poly)$ over $\mathcal{B}_1 \cup \{stCONN\}$. In order to have a corresponding non-deterministic circuit $C \in NAC^i(poly)(stCONN)$, we represent P by a set of Boolean values: the configurations that we extend to take care of additive sorts and a $box(p, q)$ relation to describe that p is associated to the additive box of the $\&$-link q. After that in the same way as [Ter04] (see section 3), the cut-elimination in P is simulated by the construction of layers of a circuit for each parallel cut-elimination step s.t. the evaluation of the constructed circuit C simulates the cut-elimination procedure. For simplicity we describe it following

the parallel reductions as in lemma 3 with the help of non-deterministic variables, one for each link of the proof net:

- The circuit simulates the choice of a slice (its edges) s.t. every distinct choice can be done in parallel, depending on the non-deterministic variables:
  - selection of one premise for each &-links, with the help of a constant depth circuit of size $O(s^3)$,
  - selection of one premise for each coad-links depending on the selected premise of the associated &-links, with the help of a constant depth circuit of size $O(s^4)$. Moreover we "erase" the coad-links themselves: by updating the *alive* value of a coad-link and the *edge* values concerning its incident edges and by linking the premise and the conclusion of a coad-link, with the help of a constant depth circuit of size $O(s^5)$,
  - selection of one component associated to each &-links using a $stCONN$ gate, with the help of a constant depth circuit of size $O(s^4)$.
- The cut between a witness (depending on the non-deterministic variables) and the modified slice is reduced in a known result due to our choice of the *id* type. This is done by a constant depth circuit of size $O(s^5)$.
- Finally as many time as $4 \times d(P)+1$ we simulate one parallel reduction of the modified slice ($\Rightarrow$, in the same way as in [Ter04] with the help of a constant depth circuit of size $O(s^3)$ that simulate $\Rightarrow_a$). We check consistency at each step with the help of a constant depth circuit of size $O(s^3)$.

The most essential cost is due to the erasing of the coad-links and the elimination of the cut between a witness and a slice of a Boolean proof net. Each small circuit used here can be found in the appendix with precise depth and size.

Theorem 8 (simulation). *Let $i \in \mathbb{N}$. For every Boolean proof net $P \in m\&BN^i(poly)$ of size s and depth d, there is a Boolean circuit in $NAC^i(poly)(stCONN)$ of size $O(s^5)$ and depth $O(d)$ which accepts the same set/language as P does.*

Since the considered non-deterministic circuit class collapse we can replace $NAC^i(poly)(stCONN)$ by $NAC^i(poly)$ in the previous theorem.

6 Conclusion

Focusing on uniformity, we strengthened the connection between proof nets and deterministic Boolean circuits by giving a uniform depth-preserving simulation of each other. This kind of Curry-Howard isomorphism for models of parallel computation as been extended to the non deterministic case. Because the uniformity arguments apply to the translation and simulation theorems, we have:

Theorem 9. *$\forall i \in \mathbb{N}$, $m\&BN^i(poly) = NAC^i(poly) = NP$.*

Corollary 2. *$m\&BN(poly) = NNC(poly) = NP$.*

The substitution on the non-deterministic Boolean type B_e deals with id to simplify the general case which can also be treated without difficulties. The most simple form of non-deterministic Boolean is $1 \oplus 1$ i.e. just replace $id = \alpha \wp \alpha^\perp$ by 1: the semantic is clearly the same but the fragment of linear logic considered is extended to neutrals. Even without this, the Boolean proof nets of type $\vdash \mathbf{B}^\perp, \ldots, \mathbf{B}^\perp, \mathbf{B} \& \ldots \& \mathbf{B}$ are in a way canonical (up to substitutions) but the simulation is more complicated. A more general work can be done using the additive fragment expressiveness fully: a garbage is no more needed but the cut-elimination is no more confluent. Other approaches are interesting like the additives à la Hughes and van Glabbeek [HvG03,HvG05]: we believe to realize a stronger seed-up.

References

[All89] Eric W. Allender. P-uniform circuit complexity. *Journal of the Association for Computing Machinery*, 36(4):912–928, 1989.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. of Comput. and System Science*, 41(3):274–306, 1990.

[BS90] R. B. Boppana and M. Sipser. *The complexity of finite functions*. MIT Press, 1990.

[CK06] S. Cook and J. Krajicek. Consequences of the provability of NP⊆P/poly, 2006.

[Dan90] V. Danos. *La logique linéaire appliquée à l'étude de divers processus de normalisation (et principalement du λ-calcul)*. PhD thesis, Univ. Paris VII, 1990.

[DR89] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28(3):181–203, 1989.

[Gir87] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987.

[HvG03] D. J. D. Hughes and R. J. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. In *Proc. IEEE Logic in Comput. Sci.*, 2003.

[HvG05] D. J. D. Hughes and R. J. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Trans. on Comput. Logic*, 2005.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th ACM Symp. on Theory of Computing*, pages 302–309, 1980.

[LdF04] O. Laurent and L. Tortora de Falco. Slicing polarized additive normalization. *Linear Logic in Computer Science*, 2004.

[MT03] H. Mairson and K. Terui. On the computational complexity of cut-elimination in linear logic. *Theoretical Computer Science*, 2841:23–36, 2003.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *J. of Computer and System Science*, 21:365–383, 1981.

[Ter04] K. Terui. Proof nets and boolean circuits. In *Proc. IEEE Logic in Comput. Sci.*, pages 182–191, 2004.

[Tor03] L. Tortora De Falco. Additives of linear logic and normalization - part i: a (restricted) church-rosser property. *T.C.S.*, 294(3):489–524, 2003.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *Siam J. Comput.*, 21(4):655–670, 1992.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, 1999.

[Wol94] Marty J. Wolf. Nondeterministic circuits, space complexity and quasigroups. *Theoretical Computer Science*, 125(2):295–313, 1994.

A Some corollaries

As a corollary for the theorem 7, we have:

Corollary 3. *For every Boolean circuit $C \in NNC^i(poly)$ of size s and depth d there is a Boolean proof net in $m\&BN(poly)$ of size $O(s^5)$ and depth $O(d)$ which accept the same set/language as C does.*

The most essential cost of the simulation of a Boolean proof net in $m\&BN^i(poly)$ ($i \in \mathbb{N}$) by a boolean circuit in $NNC^i(poly)$, is due to the translates of *stCONN* gates (a log-depth circuit of size $O(n^4)$ for a n-ary gate). Thus as a corollary for the theorem 8, we have:

Corollary 4. *For every Boolean proof net $P \in m\&BN^i(poly)$ ($i \in \mathbb{N}$) of size s and depth d, there is a Boolean circuit in $NNC^{i+2}(poly)$ of size $O(s^{12})$ and depth $O(d)$ which accepts the same set/language as P does.*

B Non-deterministic simulation

In this section, we adopt these conventions:

- $alive(p)$ is write p
- $sort(p, s)$ is write $p(s)$
- $edge(p, 0, q, i)$ is write (p, q, i)

The circuit on Fig. 7, depicts the selection of one premise of a &-links. The two ouputs are the new values for the *edge* values. This circuit is realized for all links p, q and r. For all links p and q, the new value of an $edge(p, 0, q, 1)$ or an $edge(p, 0, q, 2)$ value corresponds to the conjunction of its values at ouputs of these circuits. We obtain a circuit (named Sg) of size n^{3j} and constant depth, with n^j the number of links.

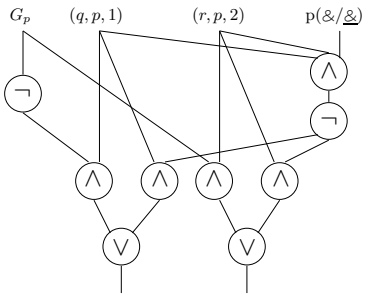


Fig. 7. selection of one premise of a &-link

It is the same for the selection of one premise of a coad-links (Fig. 8), but it depend on the selected premise of the associated &-links. The two ouputs are

the new values for the *edge* values. This circuit is realized for all links p, q, r and t. For all links p and q, the new value of an $edge(p, 0, q, 1)$ or an $edge(p, 0, q, 2)$ value corresponds to the conjunction of its values at ouputs of these circuits. We obtain a circuit of size n^{4j} and constant depth, with n^j the number of links.

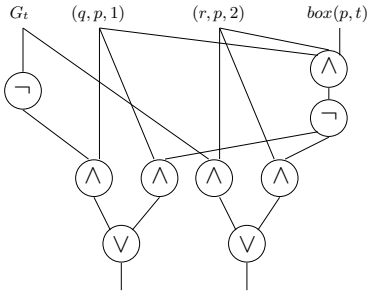


Fig. 8. selection of one premise of a coad-link

The circuit on Fig. 9, depicts the erasing of information on unary coad-links. This circuit is realized for all links p, q, r, t and i. First, the new value of a *edge* value corresponds to the disjunction of its values at the first output ($edge(q, 0, t, i)$ on the Fig. 9) of these circuits. Second, the new value of an *edge*, a *box*, or an *alive* value corresponds to the conjunction of its values at the five last outputs of these circuits and at the output just created. We obtain a circuit of size n^{5j} and constant depth, with n^j the number of links.

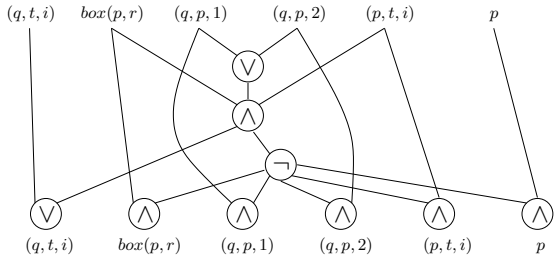


Fig. 9. erasing of a unary coad-link

The circuit on Fig. 10, depicts the erasing of the useless components of each additive box. The part above the mark -2- is realized for all links q, r and i. This circuit above the mark -3- is realized for all links p. The new value of an *edge* or an *alive* value corresponds to the conjunction of its values at ouputs of these circuits. We obtain a circuit of size n^{4j} and constant depth, with n^j the number of links.

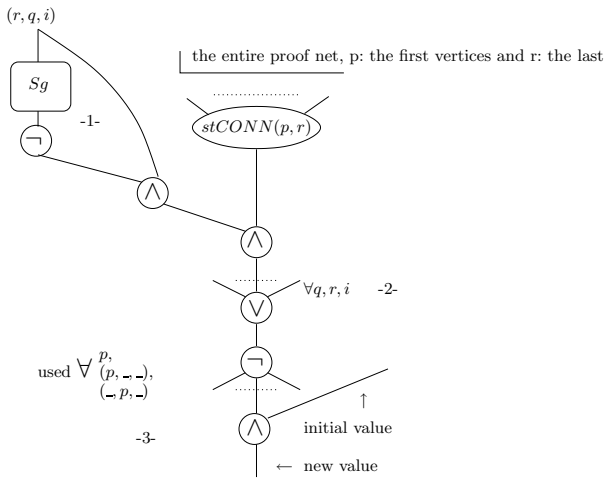


Fig. 10. erasing of a useless component in a slice

Since the cut between a witness and the modified slice is reduced in a known result, the following circuit on Fig. 11 depicts elimination of this cut. This circuit is realized for all links p, q, t, y and v. For all links p and q, the new value of an $edge(p, 0, q, 1)$ or an $edge(p, 0, q, 2)$ value corresponds to the conjunction of its values at ouputs of these circuits. For an $edge(p, 0, q, 0)$ value, we realize a disjunction. We obtain a circuit of size n^{5j} and constant depth, with n^j the number of links.

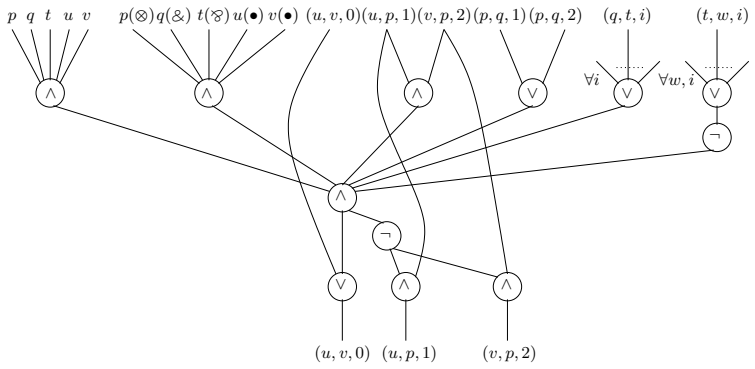


Fig. 11. fast cut-elimination of a witness cut to a boolean proof net

The circuit on Fig. 12 allows to check consistency of a slice. It is of size n^{3j} and constant depth, with n^j the number of links.

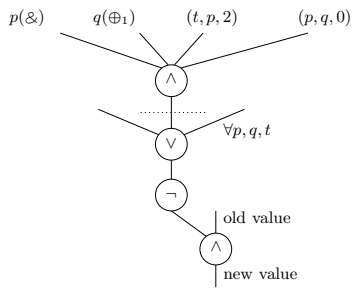


Fig. 12. check of consistency property

C Some proofs

– Descriptions of proof nets –

The description of a Boolean proof net (the triples defining a pseudo net introduced in section 3.3) can expressed as following:

Definition 1 (triple(P)). *Let $P = \{P_n\}_{n\in\mathbb{N}}$ be a family of Boolean proof nets. Links are identified by binary numbers. For all $n \in \mathbb{N}$, we fix the inputs $p_1, \ldots, p_n$ of P_n to be identified by $O, \ldots, bin(n-1)$ respectively, and the output to be identified by $bin(n)$.* triple(P) *denotes the set of tuples in $\{1\}^* \times (W \setminus \{\varepsilon\}) \times W^2$ (where W is the set of binary words), s.t. for $y = 1^n$:*

- *$\langle y, u, \epsilon, \epsilon\rangle$, u identifies a link belonging to P_n.*
- *$\langle y, u, s, \epsilon\rangle$, s identifies the sort of the link identified by u.*
- *$\langle y, u, v, bin(i)\rangle$, u identifies a link connected by its principal port (or one of its two principal ports in the case of an axiom) to the port i of the link identified by v.*

Let P_n be a Boolean proof net belonging to the family $\{P_i\}_{i\in\mathbb{N}}$. We'll show that there exists a function f computable in space $O(log(n))$, which associates $Conf(P_n)$ to 1^n iff. there exists a function g computable in space $O(log(n))$, which associate $triple(P_n)$ to 1^n. In order to do that, we will use two function h and h', both logspace in n, which associate respectively $triple(P_n)$ to $Conf(P_n)$ and $Conf(P_n)$ to $triple(P_n)$.

For every tuple belonging to its input, h write this tuple, without its last word, on its ouput, if the last word is 1.

The function h' handles the links by decreasing order on their identifiers, after having found the biggest identifier amoung the tuples $\langle \_, \_, \epsilon, \epsilon\rangle$, t. For all identifier of link $i \in \{0, \ldots, t\}$, if the tuple $\langle \_, i, \epsilon, \epsilon\rangle$ is in the input of h', it copy out on its output with the fifth word 1, else with 0. It works similarly for the tuples whose the forth word is ϵ (or none of the four).

Functions h and h' memorize a constant number of tuples, of identifier of links and of counters of size logspace in n.

– Proof of lemma 1 –

Proof. We don't want to memorize the output of f on a input w. We have to notice that at each step of the computation of g on $f(w)$, it's necessary to read only one bit of $f(w)$. Thus, it's sufficient to maintain a counter (on the binary base) of size $O(log(n))$ (where n is the size of w), because of the polynomial size of the output of f. Then, we simulate the computation of g, in space $O(log(n))$, on the input $f(w)$ without write it entirely. Each time we need to read a bit, we use a space $O(log(n))$ to find it. Hence, we use a space $O(log(n))$ to compute $g \circ f$. □

– Proof of theorem 6 –

Proof. As in a multiplicative framework, only elimination of ax-cuts cannot be performed in parallel in a slice of a proof-net built with multiplicative and additive links. Hence, we'll show that the sequence $\Rightarrow_t; \Rightarrow_{ax}; \Rightarrow_a; \Rightarrow_m$ decrease the depth a slice of a proof-net. Let P be a proof-net and $sl(P)$ one of its consistent slice.

- Let Q such that $sl(P) \Rightarrow_t Q$. If the only one cuts of $sl(P)$ are some cuts of ax-sequences, they are all removed. Then, the depth of Q is 0 which if less or equal to the depth of $sl(P)$. Else, there exist some other cuts in $sl(P)$. Let π be any sequentialization of $sl(P)$). Let $\{A_1, \ldots, A_k\}$ (where $k \in \mathbb{N}$), the set of ax-sequences of $sl(P)$. For all $i \in \{1, \ldots, k\}$, the formulae (of depth p_i) associates to the ports of the axiom which substitute the ax-sequence A_i are the same than those of any axiom in this ax-sequence. Let $p = max(p_1, \ldots, p_k)$. After reduction of all the ax-sequences, the depths of the cut-formulae (which are not between two axioms) in π remain unchanged. If the maximal depth (q) of cuts (we define the depth of a cut as the depth of formulae associated to the ports of the links connected by this cut) in $sl(P)$, which are not cuts between two axioms, is less than p, then the depth of Q is $q < p = d(sl(P))$. Else, the depth of Q is the same than the one of $sl(P)$: $q \geq p$.
- Let R such that $Q \Rightarrow_{Ax} R$. If the only one cuts of Q are ax-cuts, they are all removed. Then, the depth of R is 0 which if less than the depth of Q (since these cuts are between two links whose one is not an axiom, see below). Else, there exist some other cuts in Q. Let π be any sequentialization of Q. Let $\{v_1, \ldots, v_k\}$ (where $k \in \mathbb{N}$), the set of axioms which are cut to other links than an axiom in Q. For all $i \in \{1, \ldots, k\}$, let v_i^1 and v_i^2 be the two principal ports of v_i. Let v_i cut by its port v_i^1 to the principal port u_i^0 of a link u_i (its only one principal port because u_i cannot be an axiom). Let w_i the link connected to v_i by v_i^2. In π, the formula associated to u_i^0 is the same than the one associated to v_i^2, the dual of the one associated to v_i^1. So, they are the same depth p_i. Let $p = max(p_1, \ldots, p_k)$. The elimination of this cut remove, for all $i \in \{1, \ldots, k\}$, v_i and connect w_i to u_i. So, after elimination of all the ax-cuts, the depths of cut-formulae (those which are not ax-cuts) in π remain unchanged. If the maximal depth (q) of cuts in Q, which are not

ax-cuts, is less than p, then the depth of R is $q < p = d(Q)$. Else, the depth of R is the same than the one of Q: $q \geq p$.

– R do not contain cuts involving axioms any more. The only one cuts of R are m-cuts or additive cuts.
  - Let S such that $R \Rightarrow_a S$. Let R contain m m-cuts and n additive cuts. Let π be any sequentialization of R. An additive cut between two additive links, in a consistent slice of a proof net, whose the formulae associated to their principal ports in π are $\&_i(A)$ and $\oplus_i(B, C)$ $(i \in \{1, 2\})$ is replaced by a cut between two links whose the formulae associated to the principal ports connected by the cut in π are A and B or A and C. The deepest formulae in π (p be the greatest depth) are formulae associate to m-cuts or formulae $F_1 \in \{\&_{i_1}(A_1), \oplus_1(B_1, C_1)\}, \ldots, F_k \in \{\&_{i_k}(A_k), \oplus_{i_k}(B_k, C_k)\}$, where $0 \leq k \leq n$ and $\forall j \in \{1, \ldots, k\}, i_j \in \{1, 2\}$. The set of deepest cut-formulae contain now the formulae A_i, B_i or A_i, C_i, for all $i \in \{1, \ldots, k\}$), of depth $p-1$, which are cut-formulae. In π, an additive cut of depth p generate cuts of depth at most $p-1$ (since $\&_i(A)$ or $\oplus_i(B, C)$ $(i \in \{1, 2\})$ are formulae of depth $d(A) + 1$). The other cuts, of depth $q < p$, generate cuts of depth at most $q - 1 < p - 1$.
  - Let T such that $S \Rightarrow_m T$. Let π be any sequentialization of S. By the previous item, the deepest cut-formulae are formulae in π, associate to m-cuts. A m-cut between two links whose the formulae associate to their principal ports in π are $\otimes(\overrightarrow{A})$ and $\parr(\overleftarrow{B})$, is replaced by n cuts between pairs of links whose formulae in π associate to ports connected by cuts are A_i and B_i, for all $i \in \{1, \ldots, n\}$ where $n \in \mathbb{N}$. The deepest cut-formulae in π (p be the greatest depth) are $F_1 \in \{\otimes^{k_1}(A_1^1, \ldots, A_1^{k_1}), \parr^{k_1}((A_1^{k_1})^\perp, \ldots, (A_1^1)^\perp)\}, \ldots, F_r \in \{\otimes^{k_r}(A_r^1, \ldots, A_r^{k_r}), \parr^{k_r}((A_r^{k_r})^\perp, \ldots, (A_r^1)^\perp)\}$, where $r \in \mathbb{N}$ and $\forall j \in \{1, \ldots, r\}, k_j \in \mathbb{N}$. Then, The deepest cut-formulae are some formulae $A_i^j, (A_i^j)^\perp$ where $i \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, k_r\}$, of depth $p - 1$. In π, a cut of depth p generate some cuts of depth at most $p - 1$. Moreover, there exit at least one cut at this depth since $\otimes(\overrightarrow{A})$ or $\parr(\overleftarrow{B})$ $(n \in \mathbb{N})$ are formulae of depth $max(d(A_1), \ldots, d(A_n)) + 1$, so there exist a $j \in \{1, \ldots, n\}$ such that $A_j = max(d(A_1), \ldots, d(A_n))$. The other cuts of depth $q < p$ generate cuts of depth at most $q - 1 < p - 1$.

□