

Stratified Nested Linear Logic (short paper)

Matthieu Perrinel

ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)

matthieu.perrinel@ens-lyon.fr

Implicit computational complexity (ICC) is the field aiming to characterize complexity classes by syntactic restrictions on models of computation. For example, Light Linear Logic (*LLL*) is a linear logic subsystem characterizing *Ptime*. Finding more expressive characterizations of *Ptime* is a big challenge in ICC. This led, for instance, to L^4 and *MS*, which are orthogonal extensions of *LLL*.

Here, we define a linear logic subsystem: Stratified Nested Linear Logic (*SNLL*) which extends *LLL*, L^4 and *MS*. This system is not just the union of those systems as it can type λ -terms which are typed by none of the above systems.

1 Introduction

Implicit computational complexity is the field aiming to characterize complexity classes by syntactically restricting models of computation. The main application is to achieve automated certification of a program's complexity. Due to its focus on resources management, linear logic (*LL*) [8] is a promising setting for ICC. Through the proofs-as-programs correspondence, a *LL* subsystem characterizing *Ptime* may be transformed into a type system for the λ -calculus ensuring a polynomial bound on β reduction [2, 6].

Note that it is harder to control complexity in a higher order than in a first-order functional programming language. Thus, variants of *LL* for *Ptime* have to enforce very strict control. Consequently, some natural polynomial time λ -terms can not be typed in those variants. For example, the λ -calculus type-system *DLAL* [2] (obtained from the *LL* subsystem *LLL* [9]) is *Ptime extensionally complete*. It means that for any function f computable in polynomial time, there exists a λ -term computing f typable in *DLAL*. However, *DLAL* is not *intensionally complete*: there are λ -terms which compute in polynomial time but are not typable in *DLAL*. As a simple example, the λ -term $\lambda m. \lambda n. \lambda f. ((m)n)f$ computes multiplication of Church integers in polynomial time but is not typable in *DLAL*¹.

Exploring systems more expressive *intensionally*, i.e. which validate more *Ptime* programs, is an important challenge in ICC. This is one of the motivations behind L^4 [1] and *MS* [13], two orthogonal extensions of *LLL*.

In linear logic, programs are proofs and program execution is done by the elimination of the cut rule in the proof. Proofs are either presented as sequent calculus derivations or as *proof-nets*, a graph based syntax for proofs. In a previous work, we defined two semantic criteria on proof-nets: *stratification* and *dependence control* entailing strong *Ptime* complexity bounds. We proved that *LLL*, L^4 and *MS* (we will refer to those 3 systems as the *light logics*) verified our criteria, providing new proofs of *Ptime* soundness of those systems. Moreover, observing our criteria, we noticed that there was room for expansion: some proof-nets satisfied our criteria without being in any of the above systems.

Contributions In this paper, we define Stratified Nested Linear Logic, a subsystem of *LL*. We prove that every proof-net of *SNLL* satisfies our criteria—in fact it was designed to be as close as possible

¹To be precise, it is typable but with a type which does not allow applying the term to Church integers.

to those criteria—thus *SNLL* is *Ptime* sound. Moreover, we encode L^4 in *SNLL* (encoding of *MS* is sketched) proving *SNLL* extensional completeness. We also exhibit λ -terms which are typable in *SNLL* and not in *LLL*, L^4 and *MS*.

Related works There have been several characterizations of *Ptime* by *LL* subsystems besides the light logics. Soft Linear Logic [11] is a system which is usually considered less expressive than *LLL* even if they are orthogonal. Bounded logics (*BLL* [10] and *QBAL* [4]) are rather expressive. However, type checking in those systems is undecidable.

Similarly, Dal Lago and Gaboardi have defined the type system *dlPCF* [3] which characterizes exactly the execution time of *PCF* programs. Type-checking in *dlPCF* is undecidable, but one can imagine restricting *dlPCF* to a decidable fragment. Their framework can be seen as a top-down approach. Here we follow a bottom-up approach: we take inspiration from previous decidable type systems characterizing *Ptime* and relax conditions without losing neither soundness nor decidability.

Stratification and dependence control The notion of stratification we defined in [12] comes from Girard’s *LLL* (although he did not use the word “stratification”). “Stratification” was first used with this meaning by Danos and Joinet in [5]. Baillot and Mazza generalized this concept with the L^4 system. Our definition generalises this definition. The concept of “dependence control” is not new, but we are the first to name this concept. In *LLL* and L^4 , dependence control is enforced by requiring boxes to have at most one auxiliary door. Roversi and Vercelli generalized it with *MS*. Definition 4.3 is a generalization of the *MS*.

2 Stratified Nested Linear Logic

2.1 Intuitions on stratification and dependence control

In [12], we defined two relations $\succ_2$ and $\rightarrow$ on subterms of proof-nets such that, if the two relations are acyclic then the proof-net reduces in polynomial time. To motivate the criteria, we first state similar criteria on λ -calculus with pairs, the system whose terms are generated by $\Lambda = x \mid \lambda \langle x_1, \dots, x_k \rangle. \Lambda \mid (\Lambda)\Lambda \mid \langle \Lambda, \Lambda \rangle$ where $x, x_1, \dots, x_k$ range over a countable set of variables. Reduction is the contextual closure of $(\lambda \langle x_1, \dots, x_k \rangle. t) \langle t_1, \dots, t_k \rangle \rightarrow_\beta t[t_1/x_1; \dots; t_k/x_k]$. The $\succ_2$ and $\rightarrow$ relations will be formally defined on proof-nets in Section 4.

We define *hole-terms* as λ -terms with a variable $\circ$ appearing free exactly once². If $t \in \Lambda$, $h[t]$ denotes $h[t/\circ]$ and, if t is not a variable³, t is said a *subterm* of $h[t]$. For $t \in \Lambda$, $|t|$ is the size of t , if X is a set, $|X|$ is the cardinal of X .

If $g[u] \rightarrow_\beta v$, the *residues* of u are the copies by β -reduction of u where free variables may have been substituted. Let us consider $t = (\lambda x. \langle x, \lambda y. x \rangle) \lambda z. z \rightarrow_\beta \langle \lambda z. z, \lambda y. \lambda z. z \rangle = t'$, then the residues of $\lambda z. z$ are the two occurrences of $\lambda z. z$ in t' . The residue of $\lambda y. x$ is $\lambda y. \lambda z. z$. Finally t and $\lambda x. \langle x, \lambda y. x \rangle$ have no residue.

The non-terminating λ -term $\Omega = (\lambda x. (x)x) \lambda y. (y)y$ requires the identification of a function and its argument. In [1], Baillot and Mazza describe stratification as a way of forbidding the identification of two variables belonging to two morally different strata. Let u, v be subterms of t , we capture “ u belongs to a higher stratum than v ” by the following relation $u \rightarrow v$.

²Hole-terms are usually named “contexts”, but this word will have another meaning.

³This unusual condition is needed for the proof of Theorem 2.1.

Definition Let $t \in \Lambda$ and u, v be subterms of t , $u \rightarrow v$ if $t \rightarrow_{\beta}^* t'$ and there exists a subterm of t' of shape $(v')h[u']$ with u', v' residues of u, v .

A λ -term is said *stratified* if $\rightarrow$ is acyclic. Thus, Ω is not stratified because $\Omega \rightarrow_{\beta} (\lambda y.(y)y)\lambda y.(y)y$ so $\lambda y.(y)y \rightarrow \lambda y.(y)y$. For any stratified term $h[t_0]$, we define the *stratum* $s(t_0)$ of t_0 as its depth in terms of $\rightarrow$ (i.e. the maximum n such that $t_0 \rightarrow \dots \rightarrow t_n$). We could prove that any stratified λ -term t reduces in at most $e(|t|)$ with e a tower of exponentials whose height only depends on $s(t)$. We need an additional constraint to entail a polynomial bound.

For example, $\lambda x_1.(\lambda x_2.(\dots \lambda x_k.(\lambda z.z)\langle x_k, x_k \rangle \dots)\langle x_2, x_2 \rangle)\langle x_1, x_1 \rangle$ normalizes in time 2^k even though it is stratified (with a maximum stratum 1, independent of k). There seems to be some dependence between $\langle x_i, x_i \rangle$ and $\langle x_{i+1}, x_{i+1} \rangle$ because increasing the size of the first increases non-linearly the size of residues of the latter. This dependence is not captured by $\rightarrow$, we capture it by $\succcurlyeq_2$.

Definition Let $t \in \Lambda$ and u, v be subterms of t , $u \succcurlyeq_2 v$ if $t \rightarrow_{\beta}^* t'$ and there exists a subterm of t' of shape $(\lambda x.g[v'])h[u']$ with u', v' residues of u, v in t' , and x free at least twice in v' .

A λ -term is said to *control dependence* if $\succcurlyeq_2$ is acyclic. In this case the *nest* $n(t_0)$ of a subterm t_0 is its depth in terms of $\succcurlyeq_2$. We can notice that $\lambda x_1.(\lambda x_2.(\dots \lambda x_k.(\lambda z.z)\langle x_k, x_k \rangle \dots)\langle x_2, x_2 \rangle)\langle x_1, x_1 \rangle$ has a maximum nest of k because for every i , $\langle x_i, x_i \rangle \succcurlyeq_2 \langle x_{i+1}, x_{i+1} \rangle$.

Theorem 2.1 *If $t \in \Lambda$ is stratified and controls dependence, then there is a normalization sequence for t of length $\leq p(|t|)$ with p a polynomial which only depends on the depths of $\rightarrow$ and $\succcurlyeq_2$.*

sketch We consider the set R of subterms u of t such that there exists a residue of u which is the left part of a redex. For $u \in R$, we write $s_R(u)$ and $n_R(u)$ the depth of u in terms of $\rightarrow$ and $\succcurlyeq_2$ restricted to R . We write $S_R = \max_{u \in R} s_R(u)$, $N_R = \max_{u \in R} n_R(u)$ and $R_0 = \{u \in R \mid s_R(u) = 0\}$.

We reduce t to a term t' by reducing the redexes whose left part are in R_0 . The strategy is to reduce the outermost element of R_0 (or its residue). We can show that it is always the left part of a redex. The elements of R_0 are never in the right-part of a redex so have at most one residue. Thus $t \rightarrow^* t'$ in at most $|R_0|$ steps and $|R_0| \leq |t|$. We can show by induction on $n_R(u)$ that the number of residues of a subterm u of t is at most $|t|^{2^{n_R(u)}}$. Thus $|t'| \leq |t|^{1+2^{N_R}}$.

Let us notice that, if R' is the set of subterms of t' which have a residue which is the left part of a redex, $S_{R'} < S_R$. Thus, in S_R such rounds, we reach the normal form. Thus, we normalize $|t|$ in at most $\sum_{i=0}^{S_R} |t|^{2^{i+N_R}}$ steps.

In a previous work, we defined the notions of stratification and dependence control on *LL*. Linear logic handling duplications more precisely, the criteria on *LL* are weaker so more expressive (cf Section 4.3). The precision of *LL* on duplication also makes it easier to enforce stratification and dependence control by syntactic means, which is the object of this paper.

2.2 The formulae of Stratified Nested Linear Logic

LL [8] can be thought as a refinement of System F [7] focusing on the duplication of arguments. In *LL*, $A \Rightarrow B$ is decomposed into $!A \multimap B$. $!A$ means “infinitely many proofs of A ” and $A \multimap B$ means “using one proof of A , I can prove B ”. The conjunction is written $\otimes$ and $\forall$ allows us to quantify over the sets of formulae.

In *SNLL*, to enforce some kind of stratification and dependence control, we label the modal formulae with integers. A positive (resp. negative) occurrence of $!_{s,n}$ in a formula intuitively means that this

connective is created in a subterm of stratum $\geq s$ (resp. $\leq s$) and nest $\geq n$ (resp. $\leq n$). Variables are labelled with integers, $\forall X_s.A$ means that in the formulae replacing X_s , every left index is $\geq s$. The set of formulae is written $\mathcal{F}_0$, as defined below.

Definition For $s \in \mathbb{N}$, we define $\mathcal{F}_s$ by the following grammar (with $n, t \in \mathbb{N}$, $t \geq s$ and X ranges over a countable set of variables). Notice that $\mathcal{F}_0 \supseteq \mathcal{F}_1 \supseteq \dots$

$$\mathcal{F}_s := X_t \mid \mathcal{F}_s \multimap \mathcal{F}_s \mid \mathcal{F}_s \otimes \mathcal{F}_s \mid \forall X_s. \mathcal{F}_s \mid !_t \mathcal{F}_{t+1}$$

The left indices s' on $!$ and the indices on variables are called *stratum indices*, the right indices n on $!$ are called *nest indices*. For $A \in \mathcal{F}_0$, $s_A^{\min}$ refers to the minimum stratum index in A . We define the set H^+ (resp H^-) of positive (resp. negative) *hole-formulae* as formulae with a special variable $\circ$ appearing free exactly once, this occurrence being in a positive (resp. negative) position, i.e if there is an even (resp. odd) number of $\multimap$ containing $\circ$ in their left branch.

To gain expressivity, we define a subtyping relation $\leq$ on $\mathcal{F}_0$. The relation $\leq$, defined as the transitive closure of the following $\leq^1$ relation, follows the intuition that a connective $!_{s,n}$ in a formula means that this connective comes from a subterm of stratum $\geq s$ and nest $\geq n$.

$$A \leq^1 B \Leftrightarrow \begin{cases} \text{Either} & \exists C \in H^+, A = C[!_{s,n}D], B = C[!_{s',n'}D], s \geq s' \text{ and } n \geq n' \\ \text{Or} & \exists C \in H^-, A = C[!_{s,n}D], B = C[!_{s',n'}D], s \leq s' \text{ and } n \leq n' \end{cases}$$

We present *SNLL* in two ways: as a type-system for Λ and as a set of proof-nets built inductively. The presentation as a type system is more intuitive for many people. However, *Ptime* soundness does not stand for β -reduction but only for proof-nets, which can be thought of as generalized type-derivations.

2.3 SNLL as a sequent calculus

The typing rules of *SNLL* are presented in Figure 1. We consider a type derivation valid only if all its formulae are in $\mathcal{F}_0$. This property, along with the $\forall_e$ rule, are crucial to ensure the stratification of *SNLL*. The crucial rules for dependence control are *N* and *P*.

Let us represent any $k \in \mathbb{N}$ by its Church encoding $\underline{k} = \lambda f. \lambda x. (f)(f) \dots (f)x$ which applies k times a function to its argument. In full linear logic, one can type $\underline{k}$ with type $\mathbf{N} = \forall X. !(X \multimap X) \multimap !(X \multimap X)$. This type allows an exponential blow-up. Indeed, let us define $(+2) = \lambda m. \lambda f. \lambda x. ((m)f)(f)(f)x : \mathbf{N} \multimap \mathbf{N}$. Then we define $(*2)$ as the iteration of $(+2)$ (i.e. $(*2) = \lambda m. ((m)(+2))0 : !\mathbf{N} \multimap !\mathbf{N}$) mapping $\underline{k}$ to $\underline{k+k}$. Finally, we can type $(2\wedge) = \lambda m. ((m)(*2))1$ which maps $\underline{k}$ to $\underline{2^k}$, which is obviously not in *Ptime*: the Church encoding is unary so $\underline{2^k}$ is exponentially bigger than $\underline{k}$.

In *SNLL*, for any $s \geq 1$, and $k \in \mathbb{N}$, $\underline{k}$ can be typed with (see Figure 2 for the type derivation of $\underline{2}$):

$$\mathbf{N}_s = \forall X_s. !_{s-1,1}(X_s \multimap X_s) \multimap !_{s-1,0}(X_s \multimap X_s)$$

Let us observe how the typing constraints prevent us from typing $(2\wedge)$. Typing $(2\wedge)$ requires to type $(*2)$ by $\mathbf{N}_s \multimap \mathbf{N}_s$ for some s (we will write $\mathbf{F}_s = \mathbf{N}_s \multimap \mathbf{N}_s$). So, inside $(*2)$, $(n)(+2)$ must be typed by $\mathbf{F}_s$ with context $n : \mathbf{N}_s$. This is impossible. For example in the following attempt of type derivation, the $\forall_e$ rule replaces $\mathbf{N}_s$ by $!_{t,1}\mathbf{F}_s \multimap !_{t,0}$ so $t = s - 1$, however the formula $!_{t,0}\mathbf{F}_s$ appears in the derivation so we must have $t < s - 1$. The shape of types forbids us to type $(2\wedge)$, which is not stratified.

$$\frac{\frac{x : \mathbf{F}_s \vdash x : \mathbf{F}_s}{x : !_{t,0}\mathbf{F}_s \vdash x : \mathbf{F}_s} \quad \frac{\frac{n : !_{t,1}\mathbf{F}_s \multimap !_{t,0}\mathbf{F}_s \vdash n : !_{t,1}\mathbf{F}_s \multimap !_{t,0}\mathbf{F}_s}{n : \mathbf{N}_s \vdash n : !_{t,1}\mathbf{F}_s \multimap !_{t,0}\mathbf{F}_s} \forall_e \quad \frac{\vdash (+2) : \mathbf{F}_s}{\vdash (+2) : !_{t,1}\mathbf{F}_s}}{n : \mathbf{N}_s \vdash (n)(+2) : !_{t,0}\mathbf{F}_s} !_e \quad \frac{}{n : \mathbf{N}_s \vdash (n)(+2) : (\mathbf{N}_s \multimap \mathbf{N}_s) = \mathbf{F}_s}$$

$$\begin{array}{c}
\frac{}{x : A \vdash x : A} ax \qquad \frac{\Gamma, x_1 : A_1, \dots, x_k : A_k \vdash t : B}{\Gamma \vdash \lambda \langle x_1, \dots, x_k \rangle. t : (A_1 \otimes \dots \otimes A_k) \multimap B} \multimap_i \\
\\
\frac{\Gamma \vdash t : A \quad A \leq A'}{\Gamma \vdash t : A'} \leq \qquad \frac{\Gamma_1 \vdash t_1 : A_1 \quad \dots \quad \Gamma_k \vdash t_k : A_k}{\Gamma_1, \dots, \Gamma_k \vdash \langle t_1, \dots, t_k \rangle : A_1 \otimes \dots \otimes A_k} \otimes_i \\
\\
\frac{\Gamma, x : A \vdash t : B \quad A' \leq A}{\Gamma, x : A' \vdash t : B} \geq \qquad \frac{\Gamma \vdash t : A \multimap B \quad \Delta \vdash u : A}{\Gamma, \Delta \vdash (t)u : B} \multimap_e \\
\\
\frac{\Gamma \vdash t : A \quad X_s \text{ not free in } \Gamma}{\Gamma \vdash t : \forall X_s. A} \forall_i \qquad \frac{\Gamma, x : A[C/X_s] \vdash t : B \quad s_C^{min} \geq s}{\Gamma, x : \forall X_s. A \vdash t : B} \forall_e \\
\\
\frac{\Gamma}{\Gamma, x : !_{s,n} A \vdash t : B} W \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : !_{s,n} A \vdash t : B} D \qquad \frac{\Gamma, y : !_{s,n} A, z : !_{s,n} A \vdash t : B}{\Gamma, x : !_{s,n} A \vdash t[x/y; x/z] : B} C \\
\\
\frac{\Gamma, x : !_{s,n} !_{s+1,n+1} A \vdash t : B}{\Gamma, x : !_{s,n+1} A \vdash t : B} N \qquad \frac{\Gamma \vdash u : !_{s,n} A \quad \Delta, x : !_{s,n} A \vdash h[x] : B}{\Gamma, \Delta \vdash h[u] : B} !_e \\
\\
\frac{x_1 : A_1, x_2 : A_2, \dots, x_k : A_k \vdash t : B}{x_1 : !_{s_1,n} A_1, x_2 : !_{s_2,n+1} A_2, \dots, x_k : !_{s_k,n+1} A_k, \vdash t : !_{s,n} B} P
\end{array}$$

Figure 1: SNLL as a λ -calculus type-system.

$$\begin{array}{c}
\frac{}{g : X_s \multimap X_s \vdash g : X_s \multimap X_s} ax \qquad \frac{\frac{h : X_s \multimap X_s \vdash h : X_s \multimap X_s}{h : X_s \multimap X_s, a : X_s \vdash (h)a : X_s} ax \quad \frac{}{a : X_s \vdash a : X_s} ax}{h : X_s \multimap X_s, a : X_s \vdash (h)a : X_s} \multimap_e \\
\\
\frac{g : X_s \multimap X_s, h : X_s \multimap X_s, a : X_s \vdash (g)(h)a : X_s}{g : X_s \multimap X_s, h : X_s \multimap X_s \vdash \lambda a. (g)(h)a : X_s \multimap X_s} \multimap_i \\
\\
\frac{g : !_{s-1,0}(X_s \multimap X_s), h : !_{s-1,1}(X_s \multimap X_s) \vdash \lambda a. (g)(h)a : !_{s-1,0}(X_s \multimap X_s)}{g : !_{s-1,1}(X_s \multimap X_s), h : !_{s-1,1}(X_s \multimap X_s) \vdash \lambda a. (g)(h)a : !_{s-1,0}(X_s \multimap X_s)} \geq \\
\\
\frac{g : !_{s-1,1}(X_s \multimap X_s), h : !_{s-1,1}(X_s \multimap X_s) \vdash \lambda a. (g)(h)a : !_{s-1,0}(X_s \multimap X_s)}{f : !_{s-1,1}(X_s \multimap X_s) \vdash \lambda a. (f)(f)a : !_{s-1,0}(X_s \multimap X_s)} C \\
\\
\frac{f : !_{s-1,1}(X_s \multimap X_s) \vdash \lambda a. (f)(f)a : !_{s-1,0}(X_s \multimap X_s)}{\vdash \lambda f. \lambda a. (f)(f)a : !_{s-1,1}(X_s \multimap X_s) \multimap !_{s-1,0}(X_s \multimap X_s)} \multimap_i \\
\\
\frac{\vdash \lambda f. \lambda a. (f)(f)a : !_{s-1,1}(X_s \multimap X_s) \multimap !_{s-1,0}(X_s \multimap X_s)}{\vdash \lambda f. \lambda a. (f)(f)a : \forall X_s, !_{s-1,1}(X_s \multimap X_s) \multimap !_{s-1,0}(X_s \multimap X_s)} \forall_i
\end{array}$$

Figure 2: Type derivation of $2 : \forall X_s, !_{s-1,0}(X_s \multimap X_s) \multimap !_{s-1,0}(X_s \multimap X_s)$

$$\begin{array}{c}
\frac{}{x : A^s \vdash x : A^s} ax \quad \frac{\Gamma \vdash t : A^s \quad X \text{ not free in } \Gamma}{\Gamma \vdash t : \forall X. A^s} \forall_i \quad \frac{\Gamma, x : A[C/X]^r \vdash t : B^s}{\Gamma, x : \forall X. A^r \vdash t : B^s} \forall_e \\
\\
\frac{\Gamma, x : A^s \vdash t : B^s}{\Gamma \vdash \lambda x. t : A \multimap B^s} \multimap_i \quad \frac{\Gamma \vdash t : A \multimap B^s \quad \Delta \vdash u : A^s}{\Gamma, \Delta \vdash (t)u : B^s} \multimap_e \\
\\
\frac{x : A^{r+1} \vdash t : B^{s+1}}{x : !A^r \vdash t : !B^s} P \quad \frac{\Gamma, x : A^{r+1} \vdash t : B^s}{\Gamma, x : \$A^r \vdash t : B^s} \$e \quad \frac{\Gamma, x : A^r \vdash t : B^{s+1}}{\Gamma, x : A^r \vdash t : \$B^s} \$i \\
\\
\frac{\Gamma \vdash t : B^s}{\Gamma, x : !A^r \vdash t : B^s} W \quad \frac{\Gamma, x : A^{r+1} \vdash t : B^s}{\Gamma, x : !A^r \vdash t : B^s} D \quad \frac{\Gamma, y : !A^r, z : !A^r \vdash t : B^s}{\Gamma, x : !A^r \vdash t[x/y; x/z] : B^s} C
\end{array}$$

Figure 3: L^4 as a type system for λ -calculus.

Another way to compute $\underline{k} \mapsto \underline{2^k}$ is $(2\wedge') = \lambda m.((m)(2*))1$ which iterates the term $(2*) = \lambda m. \lambda f. \lambda x. ((m)f)((m)f)x$. To type $(2\wedge')$, we need to iterate $(2*)$, so $(2*)$ must have type $!_{s-2,n}\mathbf{N}_s \multimap !_{s-2,n}\mathbf{N}_s$. This is impossible. For example in the following attempt of type derivation, the nest indices on the new $!$ in the context formulae are equal to the nest index on the new $!$ in the type of the term. The P rule forbids us to type $(2\wedge')$ which does not control dependence.

$$\frac{\displaystyle \frac{\vdots}{p : \mathbf{N}_s, q : \mathbf{N}_s \vdash \lambda f. \lambda x. ((p)f)((q)f)x : \mathbf{N}_s}}{p : !_{s-2,n}\mathbf{N}_s, q : !_{s-2,n}\mathbf{N}_s \vdash \lambda f. \lambda x. ((p)f)((q)f)x : !_{s-2,n}\mathbf{N}_s} P \quad \frac{}{m : !_{s-2,n}\mathbf{N}_s \vdash \lambda f. \lambda x. ((m)f)((m)f)x : !_{s-2,n}\mathbf{N}_s} C$$

3 Expressivity of $SNLL$

There are already many subsystems of LL characterizing $Ptime$. We argue that the interest of $SNLL$ over the previous systems is its intentional expressivity. To support our claim we show that: the light logics can easily be encoded in $SNLL$ and there are λ -terms which are typable in $SNLL$ but do not seem typable in light logics.

3.1 Encoding of L^4

The light logics can be encoded in $SNLL$, thus $SNLL$ is at least as expressive as the union of the light logics. We exhibit the encoding of L^4 in $SNLL$. The encodings of MS is only sketched.

The L^4 types are defined by $\mathcal{F}_{L^4} = X \mid \mathcal{F}_{L^4} \multimap \mathcal{F}_{L^4} \mid \forall X. \mathcal{F}_{L^4} \mid !\mathcal{F}_{L^4} \mid \$\mathcal{F}_{L^4}$. In the type derivation rules of Figure 3, formulae are indexed by an integer (called *level*) which intuitively corresponds to the stratum of the subterm.

Any L^4 type derivation d can be transformed into a $SNLL$ derivation by replacing every indexed formula A^s by a formula $A' \in \mathcal{F}_0$ obtained from A by:

- For every $!$ in A , we have $A = H[!B]$, then we replace the $!$ by $!_{\|H\|+s,0}$ with $\|H\|$ defined by induction on hole-formulae by: $\|\circ\| = 0$, $\|\forall X. H\| = \|C \multimap H\| = \|H \multimap C\| = \|H\|$, $\|!H\| = \|\$H\| = 1 + \|H\|$ (C being a formula).

- Any variable X is indexed by $\max \{ \|H\| + i | H[X]^i \text{ appears in } d \}$
- All occurrences of $\S$ are deleted.

One can easily verify that every L^4 rule becomes a valid *SNLL* rule. The $\S$ rules become trivial (it does not change the sequent).

In *MS*, the D rule does not exist. For any occurrence of formula A in a *MS* derivation, we define $\partial(A)$ as the number of P rules below it. In *MS*, the $!$ modalities are indexed by integers corresponding to their nest. Any $!_n$ connective appearing in a formula A in a *MS* derivation will be transformed into $!_{\|A\| + \partial(A), n}$.

Concerning the other *LL* subsystems capturing *Ptime*: L_0^4 [1], *SLL*, *BLL* and *QBAL* contain proof-net which are not stratified in the sense of [12], so which are not in *SNLL*. However, no λ -term typable in L_0^4 and untypable in *SNLL* has been found yet. Even if *SLL* is orthogonal to *LLL*, *SLL* is generally thought to be less expressive than *LLL*. Finally, type-checking is undecidable in *BLL* and *QBAL* so they do not fit in our goal of automatic bound inferring.

3.2 Comparison with previous light logics

We isolate four constraints that each light logic has and which *SNLL* does not have. We illustrate each constraint with an intuitive description and a λ -term which can be typed in *SNLL* but seemingly not in any other light logic because of this constraint. We set $S = \lambda m. \lambda f. \lambda x. ((m)f)(f)x$ implementing succesor $+$ $= \lambda m. \lambda n. \lambda f. \lambda x. ((m)f)((n)f)x$ implementing addition on Church integers and $x + y + z$ is a notation for $((+)((+)x)y)z$

- In light logics, in $\langle t, u \rangle$, t and u must have the same stratum indices (depth in *LLL* and *MS*, level in L^4). The term $\underline{k} \lambda \langle x, y, z \rangle. \langle x, ((x)S)\underline{0}, x + y + z \rangle$ is not typable in light logics: because the function $\lambda \langle x, y, z \rangle. \langle x, ((x)S)\underline{0}, x + y + z \rangle$ is iterated, we have $s(y) = s(((x)S)\underline{0})$ so $s(y) > s(x)$. But, because they are in the same tuple, it must be $s(y) = s(x)$. The $x + y + z$ term ensures that the stratum indices of y and x can not be modified by $\S$ modalities.
- There is no N rule in light logics and in their encodings in *SNLL*. This seems to prevent the typing of $\underline{k}(\lambda \langle v, w, x, y, z \rangle \langle w, w, x, w + x + y, ((x)(+)v)\underline{0} \rangle)$.
- Contrary to *LLL* and L^4 , one can have several variables in the context during a P rule. So, $t = \underline{k}(\lambda \langle x, y, z \rangle. \langle x, x + y, y \rangle)$ is typable in *SNLL* but not in *LLL* and L^4 . Moreover, the maximum nest of terms is not a priori bounded by the type system, so if we set $u = \lambda \langle x, y, z \rangle. \langle z, z, z \rangle$, then $(t)(u)(t) \cdots (u)t$ is typable in *SNLL* whatever the length of the chain of applications, whereas in *MS* the maximum length of such a chain is bounded.
- In light logics, there is no subtyping. For example, in L^4 , a A^i formula can not be considered as a A^{i-1} formula. The example in the first item of this list would be typable in L^4 if it was allowed to decrease the level of a formula by mean of a subtyping relation.

3.3 Going beyond *SNLL*? (work in progress)

Though *SNLL* is more expressive than light logics, it seems there still are at least two unnecessary constraints we should get rid off before being expressive enough inside linear logic. First, any subterm of a *SNLL* proof-net must be in *SNLL* so *Ptime*. This forbids *Ptime* terms such as $\lambda n. n + (\wedge 2)3$ which computes $\underline{k} \mapsto \underline{k} + 8$ in polynomial time. The second one is that, in *SNLL*, one can not iterate a function f obtained by iteration, even if f is non-size-increasing. For example, one can not type $((\underline{3})\lambda n. ((n)S)\underline{0})\underline{0}$. More importantly, it prevents to type some natural implementations of insertion sort.

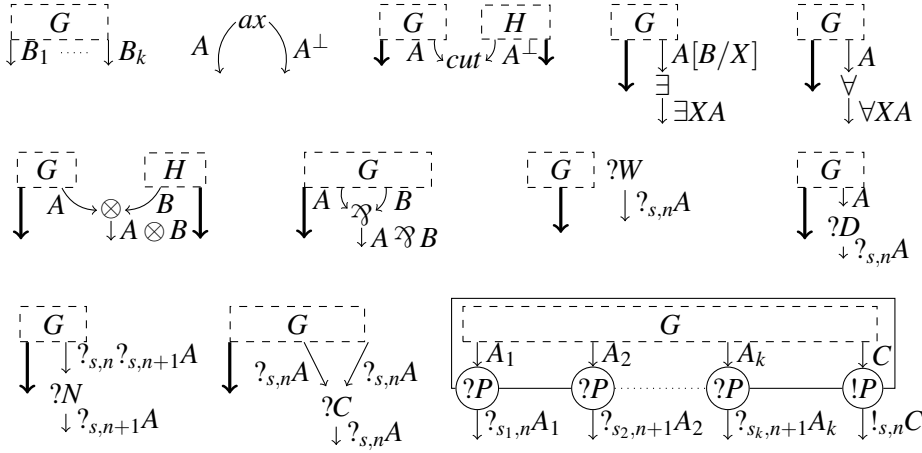


Figure 4: Construction of proof-nets. For the $\forall$ rule, we require X not to be free in the formulae labelling the other conclusions of G . For the $\exists$ rule, we require $s_B^{min} \geq s$. For the top-left rule, we suppose G is a proof-net of conclusions $A_1, \dots, A_k$ and $\forall i, A_i \leq B_i$.

A way to relax the first constraint would be to explicitly separate the function from the argument. Then, a proof-net would be said stratified (resp. controlling dependence) if there are no $\rightarrow$ (resp. $\succ_2$) cycle involving subterms in the function *and* subterms in the argument. Concerning the type system, this would mean that we would allow subterm which do not follow the indices discipline. Concretely, the stratum and nest indices would not take values in $\mathbb{N}$ but in $\mathbb{N} \cup \{\perp\}$. The $\perp$ indices would not have to follow any discipline during the typing. A term would be considered typed only if no $\perp$ appears in the conclusion.

4 Complexity properties

In this section, we will introduce *SNLL* proof-nets, which can be considered as generalizations of *SNLL* type derivations. We first presented *SNLL* as a type-system for Λ . However, the polynomial bound on execution does not hold on λ -terms, only on proof-nets. We believe that one could restrict the *SNLL* type system on Λ in the way of [2] to entail a polynomial bound on β -reduction but this is left for future work.

We will prove that the length of any sequence of *cut*-elimination for a *SNLL* proof-net G is bounded by a polynomial on the size of G . The polynomial will only depend on the maximum indices of exponential modalities and depth of G . To do this, we will use criteria based on context semantics defined in [12].

4.1 *SNLL* as a set of proof-nets

The *SNLL* typing rules on Λ correspond to the intuitionistic fragment of *SNLL*. We present the proof-nets for classical *SNLL*, as we use the results of [12] which are given in classical *LL*. We consider $A \multimap B$ as a notation for $A^\perp \wp B$. For $s \in \mathbb{N}$, we define $\mathcal{H}_s$ as (with $n, t \in \mathbb{N}$, $t \geq s$ and X ranges over a countable set of variables):

$$\mathcal{H}_s := X_t \mid X_t^\perp \mid \mathcal{H}_s \wp \mathcal{H}_s \mid \mathcal{H}_s \otimes \mathcal{H}_s \mid \exists X_t. \mathcal{H}_s \mid \forall X_t. \mathcal{H}_s \mid !_{t,n} \mathcal{H}_{t+1} \mid ?_{t,n} \mathcal{H}_{t+1}$$

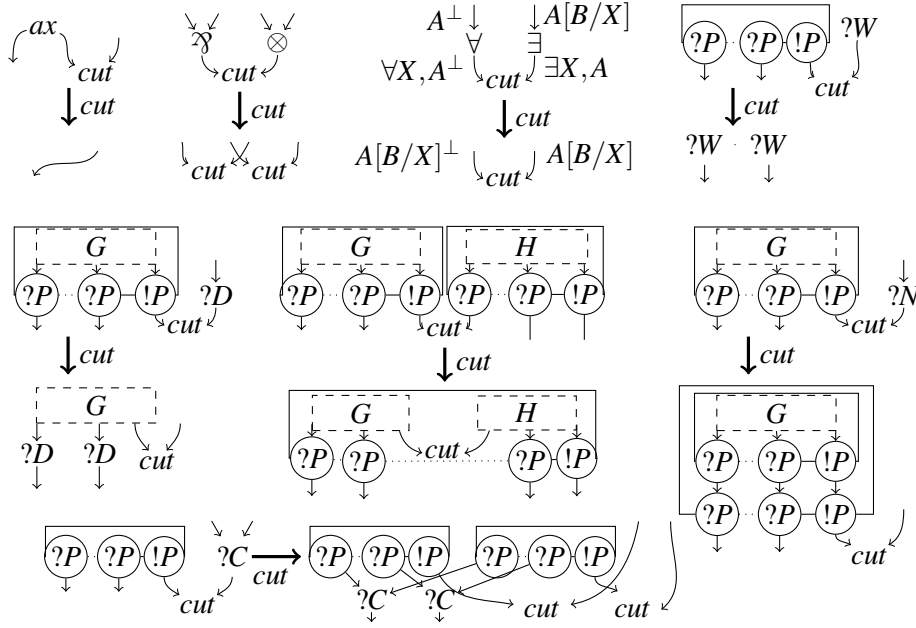


Figure 5: Rules of cut-elimination. In the $\forall/\exists$ rule, the substitution of the variable X by B takes place on the whole net.

We define inductively $(\cdot)^\perp$ on $\mathcal{K}_0$, which can be viewed as a negation: $(X_s)^\perp = X_s^\perp$, $(X_s^\perp)^\perp = X_s$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$, $(A \wp B)^\perp = A^\perp \otimes B^\perp$, $(\forall X_s. A)^\perp = \exists X_s. A^\perp$, $(\exists X_s. A)^\perp = \forall X_s. A^\perp$, $(!_{s,n} A)^\perp = ?_{s,n} (A^\perp)$ and $(?_{s,n} A)^\perp = !_{s,n} (A^\perp)$.

The $\leq$ subtyping relation is extended on $\mathcal{K}_s$ as $(\leq_1)^*$ with $\leq_1$ defined by:

$$A \leq_1 B \Leftrightarrow \begin{cases} \text{Either} & \exists C, A = C[!_{s,n} D] & B = C[!_{s',n'} D] & s \geq s', n \geq n' \\ \text{Or} & \exists C, A = C[?_{s,n} D] & B = C[?_{s',n'} D] & s \leq s', n \leq n' \end{cases}$$

SNLL proof-net A SNLL proof-net is a graph-like structure defined inductively by the graphs of Figure 4 (G and H being proof-nets). Every edge e is labelled by a $\mathcal{K}_0$ formula $\beta(e)$.

The set of edges is written E_G . The rectangle in the rightmost proof-net on the last row of Figure 4 is called a *box*. Formally a box is a subset of the nodes of the proof-net. The number of boxes containing a node n is its *depth* written $\partial(n)$. ∂_G is the maximum depth of a node of G . Let us call B the box in Figure 4. The node labelled $!P$ is the *principal door* of B , its outgoing edge is written $\sigma(B)$. The $?P$ nodes are the *auxiliary doors* of box B . The edge going out of the i -th auxiliary door is written $\sigma_i(B)$.

Proof-nets correspond, through the proofs-as-programs paradigm, to λ -terms. Intuitively, proof-nets are λ -terms where applications and abstractions are respectively replaced by $\otimes$ and $\wp$ and with additional information on duplication. cut-elimination is a relation, described in Figure 5, on proof-nets which refines β -reduction. Proof-nets are stable under cut-elimination.

4.2 Definition of contexts

The definitions of $\rightarrow$ and $\succ_2$ on Λ are based on a notion of residue. The idea of context semantics is to study the reduction of proof-nets by leaving the proof-net unchanged and analysing instead some paths

in it. The residues will be represented by *potential ports*. Cut-elimination is simulated by paths, which are defined by *contexts* travelling across the net according to a relation $\mapsto$.

The set of *signatures* is defined by: $\mathcal{S} = e \mid 1(\mathcal{S}) \mid r(\mathcal{S}) \mid p(\mathcal{S}) \mid n(\mathcal{S}, \mathcal{S})$. A signature corresponds to a list of choices of premises of $?C$ links. $r(t)$ means: “I choose the right premise, and in the next $?C$ links I will use t to make my choices”. The construction $n(t, u)$ encapsulates two sequels of choices into one. It corresponds to the $?N$ node which encapsulates two $?$ modalities into one. The $p(t)$ construction is a degenerated case of the n construction. Intuitively, $p(t)$ corresponds to $n(\emptyset, t)$. This correspondence in mind, we define an order $\sqsubseteq$ on $\mathcal{S}$ by: $n(t, u) \sqsubseteq n(t', u') \Leftrightarrow t \sqsubseteq t' \text{ and } u = u'$, $n(t, u) \sqsubseteq p(u') \Leftrightarrow u \sqsubseteq u'$, $e \sqsubseteq e$, $1(t) \sqsubseteq 1(t') \Leftrightarrow t \sqsubseteq t'$, $r(t) \sqsubseteq r(t') \Leftrightarrow t \sqsubseteq t'$ and $p(t) \sqsubseteq p(t') \Leftrightarrow t \sqsubseteq t'$.

A *potential* is a list of signatures: a signature corresponds to the duplication of one box, but an edge is duplicated whenever any of the boxes containing it is cut with a $?C$ node. For any proof-net G , Pot_G is defined as the set of pairs (e, P) with $e \in E_G$ and P is a potential of length $\partial(e)$.

Lists are written in the form $[a_1; \dots; a_n]$, $l_1 @ l_2$ represents the concatenation of l_1 and l_2 , and $.$ represents “push” $([a_1; \dots; a_n].b = [a_1; \dots; a_n; b])$.

A *trace* is a non-empty list of the following characters: $\mathcal{R}_l, \mathcal{R}_r, \otimes_l, \otimes_r, \forall, \exists, !_t, ?_t$ with t a signature. The set of traces is $\mathcal{T}$. A trace is a memory of a $\mapsto$ -path, up to cut-eliminations. We define duals of trace elements: $\mathcal{R}_l^\perp = \otimes_l, !_t^\perp = ?_t, \dots$ and extend the notion to traces by $([a_1; \dots; a_k])^\perp = [a_1^\perp; \dots; a_k^\perp]$.

A *polarity* is either $+$ or $-$. It will tell us in which way we are crossing the edges. We define $+\perp = -$ and $-\perp = +$. For $A \in \mathcal{K}_0$, $A^+ = A$ and $A^- = A^\perp$.

A *context* is a tuple (e, P, T, p) with $(e, P) \in Pot_G$, $T \in \mathcal{T}$ and p a polarity. It can be seen as a token that will travel around the net. It is located on edge e (more precisely its residue P) with orientation p and carries information T about its past travel. The nodes define two relations $\rightsquigarrow$ and $\hookrightarrow$ on contexts which are presented in Figures 6. $\mapsto$ is the union of $\rightsquigarrow$ and $\hookrightarrow$. Figure 7 shows an example of path.

4.3 Stratification and dependence control in context semantics

We define relations $\rightarrow$ and $\succcurlyeq_2$ between boxes of proof nets. They loosely correspond to the relations on Λ . In terms of context semantics paths, $B \rightarrow B'$ means that there is a path starting at the principal door of B which enters B' by its principal door, and $B \succcurlyeq_2 B'$ means that there are two paths beginning by the principal door of B which make a $\hookrightarrow$ step over B' .

$$B \rightarrow C \Leftrightarrow \exists \begin{array}{l} (\sigma(B), P) \in Pot_G, \exists t \in \mathcal{S} \\ (\sigma(C), Q) \in Pot_G, T \in \mathcal{T}, (\sigma(B), P, [!_t], +) \rightsquigarrow^* (\sigma(C), Q, T, -) \end{array}$$

$$B \succcurlyeq_2 C \Leftrightarrow \exists t \neq u, t', u' \in \mathcal{S}, \begin{cases} t \sqsubseteq t' \text{ and } (\sigma(B), P, [!_{t'}], +) \mapsto^+ (\sigma(C), Q, [!_e], +) \\ u \sqsubseteq u' \text{ and } (\sigma(B), P, [!_{u'}], +) \mapsto^+ (\sigma(C), Q, [!_e], +) \end{cases}$$

To illustrate the correspondence with the criteria on Λ , we can observe that in the first proof-net of Figure 8, $(\sigma(B_u), [], [!_e], +) \mapsto^5 (\sigma(B_v), [], [!_e; \otimes_l; ?_e], -)$ so $B_u \rightarrow B_v$. Let u, v be the λ -terms corresponding to B_u and B_v , then the whole proof-net corresponds to $(\lambda x. h[(x)u])v$ which reduces to $h[(v)u]$ so $u \rightarrow v$. In the two last proof-net of Figure 8 $B_u \succcurlyeq_2 B_v$. Indeed, in the second proof-net, we have $(\sigma(B_u), [], [!_{1(e)}], +) \mapsto^5 (\sigma(B_v), [], [!_e], +)$ and $(\sigma(B_u), [], [!_{r(e)}], +) \mapsto^5 (\sigma(B_v), [], [!_e], +)$. In the last proof-net, $n(1(e), e) \sqsubseteq p(e)$ and $n(r(e), e) \sqsubseteq p(e)$ and $(\sigma(B_u), [], [!_{p(e)}], +) \mapsto^5 (\sigma(B_v), [], [!_e], +)$. Let u, v be the λ -terms corresponding to B_u and B_v , then the whole proof-nets both correspond to $(\lambda x. h[v])u$ with x (possibly) free twice in u , so $u \succcurlyeq_2 v$.

Definition A proof-net G is stratified if $\rightarrow$ is acyclic, and it controls dependence if $\succcurlyeq_2$ is acyclic.

$$\begin{array}{c}
e \downarrow \quad \text{cut} \quad \downarrow f \qquad g \xleftarrow{ax} \quad h \qquad (e, P, T, +) \rightsquigarrow (f, P, T, -) \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad (g, P, T, -) \rightsquigarrow (h, P, T, +) \\
\\
a \searrow \quad \swarrow b \qquad e \searrow \quad \swarrow f \qquad (a, P, T, +) \rightsquigarrow (c, P, T, \mathcal{R}_l, +) \\
\quad \mathcal{R} \qquad \qquad \quad \otimes \qquad \qquad \quad (b, P, T, +) \rightsquigarrow (c, P, T, \mathcal{R}_r, +) \\
\quad \downarrow c \qquad \qquad \quad \downarrow g \qquad \qquad \quad (e, P, T, +) \rightsquigarrow (g, P, T, \otimes_l, +) \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad (f, P, T, +) \rightsquigarrow (g, P, T, \otimes_r, +) \\
\\
e \downarrow \qquad g \downarrow \qquad i \downarrow \qquad (e, P, T, +) \rightsquigarrow (f, P, T, \forall, +) \\
\quad \forall \qquad \quad \exists \qquad \quad ?N \qquad (g, P, T, +) \rightsquigarrow (h, P, T, \exists, +) \\
f \downarrow \qquad h \downarrow \qquad j \downarrow \qquad (i, P, [?_t], +) \rightsquigarrow (j, P, [?_{p(t)}], +) \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad (i, P, T, ?_t, ?_u, +) \rightsquigarrow (j, P, T, ?_{n(t,u)}, +) \\
\\
e \downarrow \qquad g \searrow \quad \swarrow h \qquad (e, P, T, +) \rightsquigarrow (f, P, T, ?_e, +) \\
\quad ?D \qquad \quad ?C \qquad \quad (g, P, T, ?_t, +) \rightsquigarrow (i, P, T, ?_{1(t)}, +) \\
f \downarrow \qquad i \downarrow \qquad \quad (h, P, T, ?_t, +) \rightsquigarrow (i, P, T, ?_{r(t)}, +) \\
\\
\left[\begin{array}{c} e \downarrow \\ ?P \end{array} \right] \text{---} \left[\begin{array}{c} g \downarrow \\ !P \end{array} \right] \qquad (e, P, T, +) \rightsquigarrow (f, P, T, ?_t, +) \\
\left[\begin{array}{c} f \downarrow \end{array} \right] \qquad \left[\begin{array}{c} h \downarrow \end{array} \right] \qquad (g, P, T, +) \rightsquigarrow (h, P, T, !_t, +) \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad (f, P, [!_t], -) \hookrightarrow (h, P, [!_t], +)
\end{array}$$

Figure 6: Context semantics rules. For any rule $(e, P, T, p) \rightsquigarrow (g, Q, U, q)$ in this figure, the dual rule $(g, Q, U^\perp, q^\perp) \rightsquigarrow (e, P, T^\perp, p^\perp)$ is also defined.

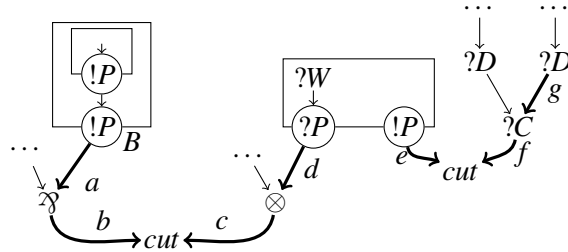


Figure 7: $(a, [], [!_{r(e)}], +) \rightsquigarrow (b, [], [!_{r(e)}; \mathcal{R}_r], +) \rightsquigarrow (c, [], [!_{r(e)}; \mathcal{R}_r], -) \rightsquigarrow (d, [], [!_{r(e)}], -) \hookrightarrow (e, [], [!_{r(e)}], +) \rightsquigarrow (f, [], [!_{r(e)}], -) \rightsquigarrow (g, [], [!_e], -)$

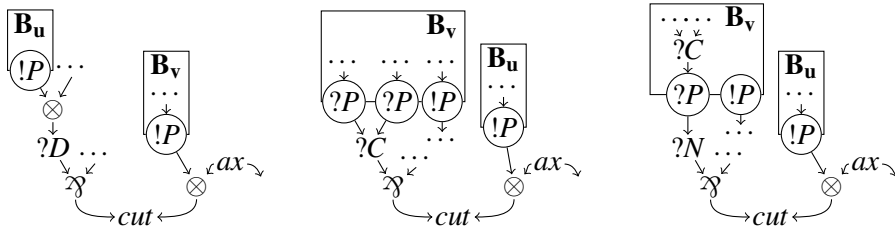


Figure 8: The first proof-net corresponds to $(\lambda x. h[(x)u])v$ for some h , the last two proof-nets correspond to $(\lambda x. h[v])u$ for some h .

The following theorem is the main result of [12].

Theorem 4.1 *The maximal reduction length of a stratified proof-net G controlling dependence (with $n = |E_G|$, S being the depth of $\multimap$ and N the depth of $\multimap_2$) is bounded by*

$$n^{(\partial_G \cdot (12N+22))^{S+1}}$$

4.4 Underlying formula of a context

The purpose of *SNLL* indices is that, if $\beta(\sigma(B)) = !_{s,n}A$ and $\beta(\sigma(B')) = !_{s',n'}B$, then: $B \multimap B' \Rightarrow s > s'$ and $B \multimap_2 B' \Rightarrow n > n'$. Stratification and dependence control follow immediately.

To prove this, we will observe that if $(\sigma(B), P, [!_t], +) \rightsquigarrow^* C$, then C has information about the formula $\beta(\sigma(B))$. We want to define an underlying formula $\beta(C)$ for any context C such that, if $C \rightsquigarrow D$, then $\beta(D) \leq \beta(C)$. Our first idea to define $\beta(e, P, [!_t]@T, +)$ would be to use T to prune the syntactic tree of $\beta(e)$. For example $\beta(e, P, [!_{1(e)}; \mathfrak{A}_r; \forall], +)$, with $\beta(e) = \forall X. ?_{s,m}(X \otimes X^\perp) \mathfrak{A} !_{t,n}(X^\perp \mathfrak{A} X)$, would be $!_{t,n}(X^\perp \mathfrak{A} X)$. Noticing that crossing *cut* and *ax* does not change the trace but transforms the labelling formula into its dual, we would define $\beta(e, P, [!_t]@T, -)$ as $\beta(e)^\perp$ pruned using T .

However, there is a problem with this definition when we cross a $\exists$ link downwards. For example, if $(c, [], [!_e; \otimes_r], +) \mapsto (d, [], [!_e; \otimes_r; \exists], +)$ with $\beta(c) = ?_{s,m}(X \otimes X^\perp) \otimes !_{t,n}(X^\perp \mathfrak{A} X)$ and $\beta(d) = \exists Y.Y$. Then, $\beta(c, [], [!_e; \otimes_r], +)$ would be $!_{t,n}(X^\perp \mathfrak{A} X)$, but $\beta(d, [], [!_e; \otimes_r; \exists], +)$ would be undefined: the trace is not compatible with the syntactic tree of $\beta(d)$. To define underlying formulae of contexts, we will pay attention to quantifiers. The precise definition of $\beta(C)$ is given below. It is quite technical. The main result of this paper can be proved and understood just with the above intuition, Definition 4.4 and Lemmas 4.2 and 4.3.

First, we have to substitute the variables associated with the $\forall$ nodes. For any $(e, [p_1; \dots; p_{\partial(e)}]) \in Pot_G$, we define $\beta(e, [p_1; \dots; p_{\partial(e)}])$ as $\beta(e)$ where the variable Z associated to the $\forall$ node of conclusion f is substituted in the following way: If $(f, [p_1; \dots; p_{\partial(f)}], [\forall], +) \mapsto^* (g, Q, [\forall], -)$ with g the conclusion of a $\exists$ link whose premise is h , then we substitute Z by $\beta(h, Q)$, otherwise Z is not substituted. The proof that this process terminates can be found in [12].

Let $C = (e, P, [!_t]@T, p)$ be a context, the underlying formula of C , $\beta(C)$, is defined as $\gamma(\beta(e, P)^p, T, [])$ with γ defined by induction on T :

- $\gamma(A, [], T') = A$
- $\gamma(A \otimes B, T, [\otimes_l]@U) = \gamma(A, T, [\otimes_l]@U)$, $\gamma(A \otimes B, T, [\otimes_r]@U) = \gamma(B, T, [\otimes_r]@U)$
- $\gamma(A \mathfrak{A} B, T, [\mathfrak{A}_l]@U) = \gamma(A, T, [\mathfrak{A}_l]@U)$, $\gamma(A \mathfrak{A} B, T, [\mathfrak{A}_r]@U) = \gamma(B, T, [\mathfrak{A}_r]@U)$
- If $(e, P, [\forall]@U, p) \mapsto^* (g, Q, [\forall], -)$ with g the conclusion of a $\exists$ link of premise h , then $\beta(g, Q) = \exists X.B$ and $\beta(h, Q) = B[C/X]$ and $\gamma(\forall X.A, T, \forall, U) = \gamma(A[C/X], T, [\forall]@U)$. Else, $\gamma(\forall X.A, T, \forall, U) = \gamma(A, T, [\forall]@U)$.
- If $(g, Q, [\exists], +) \mapsto^* (e, P, [\exists]@U, p)$ with g the conclusion of a $\exists$ link of premise h , then $\beta(g, Q) = \exists X.B$ and $\beta(h, Q) = B[C/X]$ and $\gamma(\exists X.A, T, \exists, U) = \gamma(A[C/X], T, [\exists]@U)$. Else, $\gamma(\exists X.A, T, \exists, U) = \gamma(A, T, [\exists]@U)$.
- $\gamma(!_{s,n}A, T, [!_t]@U) = \gamma(A, T, [!_t]@U)$, $\gamma(?_{s,n}A, T, [?]@U) = \gamma(A, T, [?]@U)$

Definition Let C be a context, if $\beta(C) = !_{s,n}A$ we set $s(C) = s$ and $n(C) = n$.

Lemma 4.2 *If $C \mapsto D$ and $s(C)$ and $n(C)$ are defined, then $s(D)$ and $n(D)$ are defined, $s(C) \geq s(D)$ and $n(C) \geq n(D)$.*

Lemma 4.3 If $\beta(e, P, T, p) = !_{s,m}A$ then

$$s \geq \min \{ *|t \} \beta(e)^p = H[X_t], \beta(e)^p = H[X_t^\perp] \text{ or } \beta(e)^p = H[!_{t,n}A']$$

4.5 SNLL is stratified and controls dependence

Lemma 4.4 If $B \rightarrow B'$, $\beta(\sigma(B)) = !_{s,n}A$ and $\beta(\sigma(B')) = !_{s',n'}A'$ then $s > s'$

Proof If $B \rightarrow B'$, then $C = (\sigma(B), P, [!_t], +) \rightsquigarrow^+ (\sigma(B'), P', T, !_u, -) = D$. According to Lemma 4.2, $s(C) \geq s(D)$. We know that $s(C) = s$. Moreover, $\beta(\sigma(B'))^- = ?_{s',n'}A'^\perp$ with $A' \in \mathcal{K}_{s'+1}$. So, according to Lemma 4.3, $s(D) \geq s' + 1$. Finally, $s = s(C) \geq s(D) > s'$.

Lemma 4.5 If $B \succ_2 C$, $\beta(\sigma(B)) = !_{s,n}A$ and $\beta(\sigma(C)) = !_{s',n'}A'$ then $n > n'$

Proof If $B \succ_2 C$ then there exists $t \neq u, t', u' \in \mathcal{S}$ such that $t \sqsubseteq t', u \sqsubseteq u'$, and

$$\begin{aligned} (\sigma(B), P, [!_t], +) &= C_n \mapsto C_{n-1} \mapsto \dots \mapsto C_1 = (\sigma(C), Q, [!_e], -) \\ (\sigma(B), P, [!_{u'}], +) &= C'_{n'} \mapsto C'_{n'-1} \mapsto \dots \mapsto C'_1 = (\sigma(C), Q, [!_e], -) \end{aligned}$$

Let k be the lowest index such that $C_k \neq C'_k$. Then, either:

- k does not exist. This means that $\forall k, C_k = C'_k$ so $t' = u'$. Thus, either $t \neq t'$ or $u \neq u'$. We suppose that $t \neq t'$. So there is a $p(\)$ construction inside t' which disappears along the path from C_n to C_1 . Let C_i be the first context in this path such that the left-most trace element of the trace does not contain a $p(\)$. Then $C_{i+1} = (e, R, [p(v)], -) \mapsto (f, R, [v], -) = C_i$, $\beta(C_{i+1}) = \beta(e)^\perp = !_{s_i, n_i+1}A$ and $\beta(C_i) = \beta(f)^\perp = !_{s_i, n_i}!_{s_i, n_i+1}A$. Thus, $n(C_{i+1}) = n_i + 1 > n_i = n(C_i)$ so, according to Lemma 4.2, $n = n(C_n) \geq n(C_{i+1}) > n(C_i) \geq n(C_1) = n'$.
- Else, $\rightsquigarrow$ being injective, the $C_k \mapsto C_{k-1}$ step must be $C_k \hookrightarrow C_{k-1}$

$$\begin{cases} C_k = (\sigma_j(D), P_k, [!_u], -) \hookrightarrow (\sigma(D), P_k, [!_u], +) = C_{k-1} \\ C'_k = (\sigma_{j'}(D), P_k, [!_u], -) \hookrightarrow (\sigma(D), P_k, [!_u], +) = C_{k-1} \end{cases}$$

But $C_k \neq C'_k$ so either $j \neq 1$ or $j' \neq 1$. We suppose without loss of generality, that $j \neq 1$. Then $n = n(C_n) \geq n(C_k) > n(C_{k-1}) \geq n(C_1) = n'$ so $n > n'$.

Those lemmas state that the indices correspond to the strata and nests of boxes. Then, *Ptime* soundness follows immediately.

Theorem 4.6 Let G be a SNLL proof-net, then the maximal reduction length of G (with $n = |E_G|$, S be the maximum stratum index in G and N the maximum nest index in G) is bounded by

$$n^{(\partial_G \cdot (12N+22))^{S+1}}$$

Proof By Lemma 4.4, G is stratified and the maximum stratum is $\leq S$. By Lemma 4.5, G controls dependence and the maximum nest is $\leq N$. The result follows by Theorem 4.1.

5 Conclusion

Thanks to the semantic criteria defined in [12], we defined a new characterization of *Ptime* by a *LL* subsystem. This system is another step towards expressive characterizations of *Ptime* by syntactic means.

Notice that SNLL is decidable in the following way: any SNLL proof-net G can be transformed into a SNLL proof-net whose indices are at most $|E_G|$. So, given a LL proof-net G , one can try all possible indices and verify whether there is a decoration satisfying the SNLL constraints.

References

- [1] P. Baillot and D. Mazza. Linear logic by levels and bounded time complexity. *Theoretical Computer Science*, 411(2), 2010.
- [2] P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1), 2009.
- [3] U. Dal Lago and M. Gaboardi. Linear dependent types and relative completeness. In *Logic in Computer Science, 2011*. IEEE, 2011.
- [4] U. Dal Lago and M. Hofmann. Bounded linear logic, revisited. In *Typed Lambda Calculi and Applications*. Springer, 2009.
- [5] V. Danos and J.B. Joinet. Linear logic and elementary time. *Information and Computation*, 183(1), 2003.
- [6] M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for λ -calculus. In *Proceedings of Computer Science Logic*, volume 4646, 2007.
- [7] J.Y. Girard. Une extension de l'interpretation de gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. *Studies in Logic and the Foundations of Mathematics*, 63, 1971.
- [8] J.Y. Girard. Linear logic. *Theoretical computer science*, 50(1), 1987.
- [9] J.Y. Girard. Light linear logic. *Information and Computation*, 143(2), 1998.
- [10] J.Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical computer science*, 97(1), 1992.
- [11] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2), 2004.
- [12] M. Perrinel. On paths-based criteria for polynomial time complexity in proof-nets (long version). <http://arxiv.org/abs/1201.2956>, 2013.
- [13] L. Vercelli. *On the Complexity of Stratified Logics*. PhD thesis, Scuola di Dottorato in Scienze e Alta Tecnologia, Università degli Studi di Torino – Italy, 2010.