Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 318 (2004) 163 – 180
Theoretical Computer Science
www.elsevier.com/locate/tcs

Soft linear logic and polynomial time

Yves Lafont
Fédération de Recherche des Unités de Mathématiques de Marseille, Institut de Mathématiques de Luminy, Universite de le Mediterranee, UPR 9016 du CNRS, 163 Avenue de Luminy, case 930, 13288 Marseille Cedex 9, France

Abstract

We present a subsystem of second-order linear logic with restricted rules for exponentials so that proofs correspond to polynomial time algorithms, and vice versa.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Linear logic; Second order lambda-calculus; Polynomial time

0. Introduction

Most computational systems based on the paradigm of cut elimination, such as *Gödel's system T* and *Girard's system F*, are very expressive: they contain at least primitive recursion. On the other side, the multiplicative additive fragment of linear logic has the property that any proof reduces to a cut-free one in a linear number of steps. It is quite natural to look for an intermediate system corresponding to polynomial time computation. The following proposals are based on second-order logic.

- *Strictly predicative comprehension* (see [9]). This system is not based on linear logic, but on intuitionistic logic. The main problem is that it contains simply typed λ-calculus, so that there is no particular bound on the length of normalization. Furthermore, it is clear that the restriction on the comprehension scheme is a way to control the use of contraction.
- *Bounded linear logic* (see [6]). A nice system, but with a heavy syntax, since polynomials appear explicitly in types.
- *Light linear logic* (see [4]). The types are essentially those of linear logic, but the rules for exponentials are modified. The problem is that an extra modality is needed,

E-mail address: lafont@iml.univ-mrs.fr (Y. Lafont).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.10.018

as well as a complicated notion of hybrid sequent. The latter is avoided in *light affine logic* (see [1]) by adding the unrestricted rule of weakening. For that reason, light affine logic has been traditionally used in the literature instead of light linear logic in all recent works on this subject (see for instance [10]). *Elementary linear logic* (see [4]) is a variant of Light linear logic which is complete for elementary recursive computation.

Soft linear logic can be seen as a subsystem of bounded linear logic which is powerful enough to encode polynomial time. It is simpler than bounded linear logic and light linear logic. In particular, it is very easy to show that we have computation in polynomial time (Theorem 2). Furthermore, the degree of the polynomial bound is just the *exponential depth* of the proof. The fact that our system is complete for polynomial time computation (Theorem 9) is more delicate to prove, but the essential ideas are contained in the proofs of Lemmas 5 (encoding addition and multiplication) and 7 (encoding predecessor and successor). The author is grateful to Luca Roversi for a fruitful discussion that led him towards the proof of Theorem 9.

1. Soft linear logic

We start from the intuitionistic version of linear logic, with the following connectives and quantifiers: $\multimap$ (*linear implication*), $\otimes$ (*multiplicative conjunction*), $\mathbf{1}$ (*multiplicative unit*), & (*additive conjunction*), ! (*exponential modality*), and $\forall$ (*universal quantifier*). The other intuitionistic linear connectives and quantifiers are definable as follows: $A \oplus B = \forall\alpha.(A \multimap \alpha) \,\&\, (B \multimap \alpha) \multimap \alpha$, $\mathbf{0} = \forall\alpha.\alpha$, $\exists\alpha.A = \forall\beta.(\forall\alpha.A \multimap \beta) \multimap \beta$, and $\top = \exists\alpha.\alpha$. In fact, $\otimes$, $\mathbf{1}$, and & are themselves definable in terms of $\multimap$, !, and $\forall$, but it is convenient to consider them as primitive.

If A is a formula and $n \in \mathbb{N}$, we write A^n for the formula $A \otimes \cdots \otimes A$ (n times) and $A^{(n)}$ for the sequence $A, \ldots, A$ (n times). Any operation on formulas is extended to sequences of formulas in an obvious way. For instance, if Γ is a sequence of formulas $C_1, \ldots, C_n$, we write $!\Gamma$ for the sequence $!C_1, \ldots, !C_n$, and if Δ is a sequence of formulas $D_1, \ldots, D_n$ of the same length as Γ, we write $\Gamma \otimes \Delta$ for the sequence $C_1 \otimes D_1, \ldots, C_n \otimes D_n$.

In ILL$_2$ (*second-order intuitionistic linear logic*), sequents are of the form $\Gamma \vdash A$, where Γ is a sequence of formulas called the *hypotheses* and A is a formula called the *conclusion*. The rules are the following:

- *exchange* (the only structural rule), *identity*, and *cut*:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C} \qquad \frac{}{A \vdash A} \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C},$$

- multiplicative logical rules:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C},$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{}{\vdash \mathbf{1}} \qquad \frac{\Gamma \vdash C}{\Gamma, \mathbf{1} \vdash C},$$

- additive logical rules:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A\&B} \quad \frac{\Gamma, A \vdash C}{\Gamma, A\&B \vdash C} \quad \frac{\Gamma, B \vdash C}{\Gamma, A\&B \vdash C},$$

- exponential logical rules:

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A} \quad \frac{\Gamma, A \vdash C}{\Gamma, !A \vdash C} \quad \frac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C} \quad \frac{\Gamma \vdash C}{\Gamma, !A \vdash C},$$

- quantification logical rules:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall\alpha.A} \quad \frac{\Gamma, A[B/\alpha] \vdash C}{\Gamma, \forall\alpha.A \vdash C}.$$

The *logical rules* are divided into *right logical rules*, where the connective or the quantifier occurs as a conclusion, and *left logical rules*, where it occurs as a hypothesis. As usual, in the right logical rule for $\forall$, there must be no free occurrence of α in Γ.

The four exponential rules are, respectively, called *promotion*, *dereliction*, *contraction*, and *weakening*. Note that dereliction, contraction, and weakening correspond to the following axioms:

$$!A \multimap A, \quad !A \multimap !A \otimes !A, \quad !A \multimap \mathbf{1}.$$

The four exponential rules can be replaced by *soft promotion*, *digging*, and *multiplexing*:

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A} \quad \frac{\Gamma, !!A \vdash C}{\Gamma, !A \vdash C} \quad \frac{\Gamma, A^{(n)} \vdash C}{\Gamma, !A \vdash C}.$$

In multiplexing, the *rank* n can be any natural number. In particular, we get weakening for $n = 0$, and dereliction for $n = 1$. Note that digging and multiplexing correspond to the following axioms:

$$!A \multimap !!A, \quad !A \multimap A^n.$$

SLL$_2$ (*second-order soft linear logic*) is ILL$_2$ where the exponential rules have been replaced by *soft promotion* and *multiplexing* (without *digging*):

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A} \quad \frac{\Gamma, A^{(n)} \vdash C}{\Gamma, !A \vdash C}.$$

Theorem 1. SLL$_2$ *satisfies cut elimination.*

The argument is essentially the same as for ILL$_2$ or for its classical version LL$_2$ (see [3]), but with the following reductions for the exponential rules:

$$\frac{\dfrac{\Gamma \vdash A}{!\Gamma \vdash !A} \quad \dfrac{\Delta, A^{(n)} \vdash C}{\Delta, !A \vdash C}}{!\Gamma, \Delta \vdash C} \quad \rightarrow \quad \frac{\dfrac{\Gamma \vdash A \cdots \Gamma \vdash A \quad \Delta, A^{(n)} \vdash C}{\Gamma^{(n)}, \Delta \vdash C}}{!\Gamma, \Delta \vdash C},$$

$$\frac{\dfrac{\Gamma \vdash A}{!\Gamma \vdash !A} \quad \dfrac{\Delta, A \vdash C}{!\Delta, !A \vdash !C}}{!\Gamma, !\Delta \vdash !C} \quad \rightarrow \quad \frac{\dfrac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}}{!\Gamma, !\Delta \vdash !C}.$$

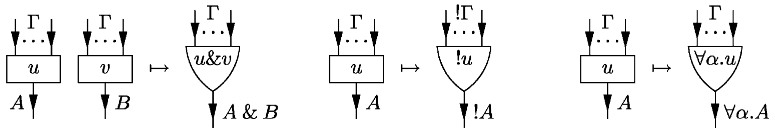
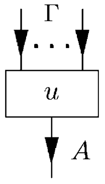


Fig. 1. Boxes.

In the right member of the first rule, we use generalized versions of cut and multiplexing, which are obviously derivable. The second rule is a commutative conversion. In fact, cut elimination is much more *feasible* in SLL_2 than in ILL_2, as we shall see in the restricted case of *external reduction*.

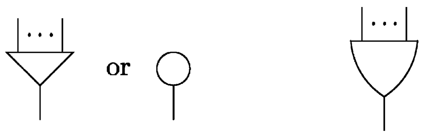
2. Proof nets

We follow essentially the conventions of [7], but in an intuitionistic framework. A *proof net* u for the sequent $\Gamma \vdash A$ consists of *cells* which are connected through oriented *wires*. Each wire is *typed* by a formula. There is one *input* for each hypothesis in Γ and one *output* for the conclusion A:



(1)

We have *atomic cells* and *compound cells* (or *boxes*) which are, respectively, pictured as follows:



(2)

In both cases, there are several *auxiliary ports* (possibly none) and one *principal port*, but the auxiliary ports are not necessarily inputs and the principal port is not necessarily an output. An atomic cell may carry some extra information, such as a boolean value 0 or 1, or a formula B. A box refers to one or two already constructed proof nets. There are three kinds of boxes corresponding to the right logical rules for &, !, and $\forall$ (Fig. 1). By definition, boxes are proof nets, and the other rules for building proof nets correspond to the remaining rules of SLL_2 (Fig. 2). Note the following points:

- Exchange allows crossing of wires. For clarity, we have chosen *planar versions* of the other rules.
- Identity means that a single wire is a proof net.

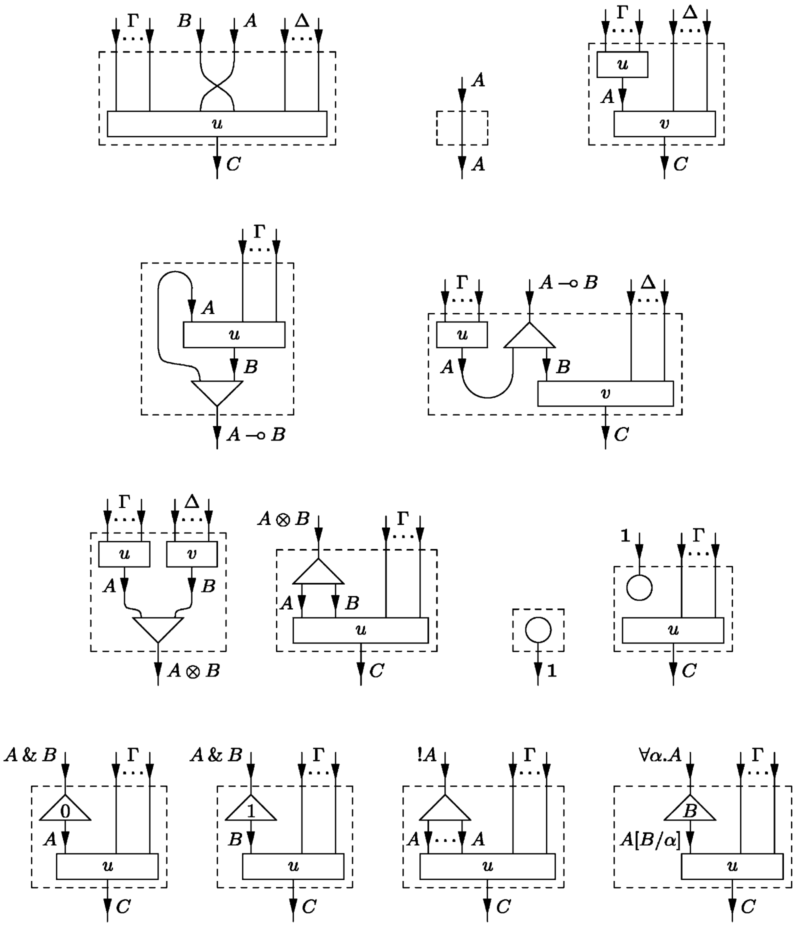


Fig. 2. Other rules for building proof nets.

- Cut consists in plugging the output of a proof net to the input of another proof net.
- Multiplexing of rank n consists in plugging a *multiplexor of rank n*, which is an atomic cell labelled by n, with n auxiliary ports.

Following the terminology of [7], cut elimination corresponds to three kinds of reduction for proof nets: *external reduction*, *internal reduction*, and *commutative reduction*. Here, we shall only consider *external reduction*, which defines an *interaction system* (Fig. 3). It is not hard to see that proof nets are closed under external reduction: it suffices to check that this reduction can always be interpreted at the level of proofs. In the absence of multiplicative units, one can also use the *Danos–Regnier criterion* (see [2] or [7]).

A proof net can be seen as an equivalence class of proofs. So we shall identify proofs with proof nets. For any proof net u, we define the *degree*, the *rank*, and the

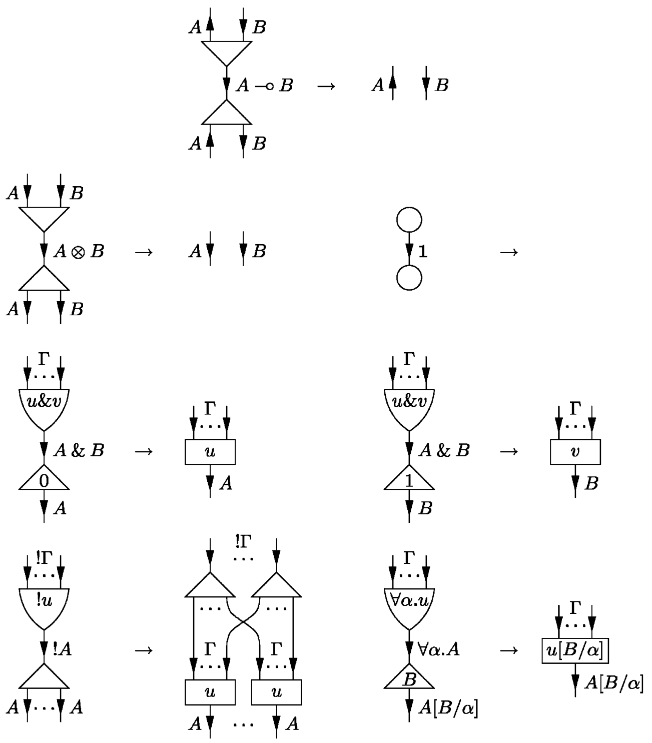


Fig. 3. External reduction for proof nets.

weight of u as follows:

- The *degree* of u counts the nesting of exponential boxes in u. It is 0 if u contains no exponential box, or $p+1$ if u contains exponential boxes of the form $!v$ and the maximal degree of such a v is p.
- The *rank* of u is the maximal rank of multiplexors in u. If all these ranks are the same, we say that u is *homogeneous*, and if there is no multiplexor, we say that u is *generic*. A generic proof net will be considered as a homogeneous proof net of rank n for any n.
- The *weight* of u is a polynomial W_u. It is the sum of the weights of all its cells, where the weight of an atomic cell is constant 1 if it corresponds to a right logical rule, or constant 0 if it corresponds to a left logical rule, and the weight of a box is given by the following formulas:

$$W_{u\&v} = W_u + W_v + 1, \quad W_{!u} = XW_u + 1, \quad W_{\forall\alpha.u} = W_u + 1.$$

Note that the weight of a multiplexor is 0, so that for any proof net u of rank n, the natural number $W_u(n)$ decreases at each reduction step, and we get the *normalization property*:

Theorem 2. *A proof net u of rank n reduces to a unique normal form in at most $W_u(n)$ steps.*

The uniqueness comes from the *confluence property*, which holds for any interaction system, because the cells may only interact through their principal ports. In fact, the computation leading to the normal form is unique up to commutation of reductions. Note also the following points:

- For any proof net u of degree p, we have $W_u(n) \leqslant kn^p$ for all $n \geqslant 1$, where $k = W_u(1)$. So we get normalization in *exponential time*.
- Typing is not used in this result. This means that normalization in exponential time holds for *untyped nets*. This also means that we can add arbitrary *fixpoints of types* (as in [4]), such as $A = !A \multimap A$, without loosing the normalization property.
- The logical rules for quantifiers are superfluous in the untyped case: one can remove the atomic cells corresponding to the left logical rule for $\forall$ and replace each box $\forall\alpha.u$ by its content u.

3. Translations

It is well known that second-order intuitionistic logic can be embedded into ILL_2 as follows: $(A \Rightarrow B)^* = !A^* \multimap B^*$, $(A \wedge B)^* = A^* \,\&\, B^*$, and $(\forall\alpha.A)^* = \forall\alpha.A^*$. There is also a translation in the opposite direction which forgets the exponentials: $(A \multimap B)_* = A_* \Rightarrow B_*$, $(A \otimes B)_* = (A \,\&\, B)_* = A_* \wedge B_*$, $(!A)_* = A_*$, and $(\forall\alpha.A)_* = \forall\alpha.A_*$.

Both translations apply to proofs and are compatible with reduction. The first translation does not give proofs in SLL_2, but of course, the second one applies to SLL_2, which is a subsystem of ILL_2. This means that each proof in SLL_2 can be interpreted in second order typed λ-calculus with explicit pairs. The interpretation of a proof net as a typed λ-term is straightforward: for instance, the atomic cells for linear implication correspond, respectively, to λ-abstraction and to application, and the multiplexor corresponds to sharing.

It is also possible to translate SLL_2 into IMALL_2, the multiplicative additive fragment of ILL_2. First, we consider the case of homogeneous proofs of rank n (including generic proofs). This translation consists in replacing $!A$ by A^n. The multiplexor is interpreted by a combination of multiplicative atomic cells, and the box $!u$ by the proof net $u^n = u \otimes \cdots \otimes u$ (n times), where the *tensor product* $u \otimes v$ of two proof nets u and v with the same number of inputs is defined by Fig. 4. This translation is compatible with reduction, but it increases the size of nets exponentially. This is not surprising since in IMALL_2, we have normalization in a linear number of steps. This translation can be adapted to the non-homogeneous case: it suffices to replace $!A$ by $(A \,\&\, \mathbf{1})^n$.

By generalizing this idea, we shall define translations of SLL_2 into itself. First, the notation A^n is extended to *polynomial expressions* as follows:

$$A^X = !A, \quad A^{P+Q} = A^P \otimes A^Q, \quad A^{PQ} = (A^P)^Q.$$

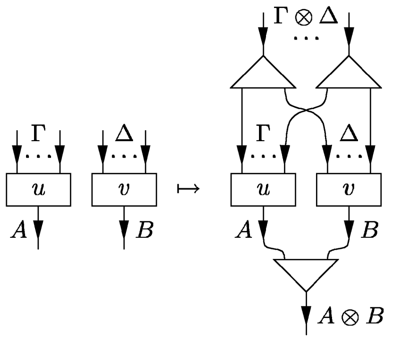


Fig. 4. Tensor product of two proof nets.

Note that the following equivalences and equalities hold:

$$A^{(P+Q)+R} \equiv A^{P+(Q+R)}, \quad A^{P+0} \equiv A^{P}, \quad A^{P+Q} \equiv A^{Q+P},$$

$$A^{(PQ)R} = A^{P(QR)}, \quad A^{1P} = A^{P} = A^{P1}, \quad A^{P(Q+R)} = A^{PQ+PR}.$$

However, the equivalences $A^{PQ} \equiv A^{QP}$ and $A^{(P+Q)R} \equiv A^{PR+QR}$ do not hold in general, and this is why we must consider polynomial expressions rather than polynomials. A *polynomial expression* is just a term built from natural numbers and a variable X, using addition and multiplication. A polynomial with coefficient in $\mathbb{N}$ is not defined by a unique polynomial expression, but it can be represented by its *Horner normal form*. For instance, the Horner normal form of the polynomial $2X^3 + 3X^2 + 5$ is the polynomial expression $5 + XX(3 + X2)$.

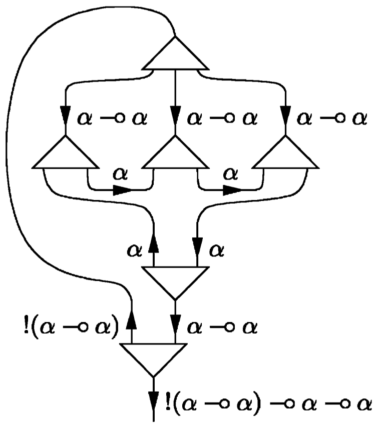
We write $A\langle P\rangle$ for the formula A where each subformula of the form $!B$ is replaced by B^P. For instance $A\langle XX\rangle$ is A where each exponential is doubled. By induction on P, it is easy to see that the following rules are derivable (*generalized soft promotion* and *generalized multiplexing*):

$$\frac{\Gamma \vdash A}{\Gamma^P \vdash A^P} \quad \frac{\Gamma, A^{(P(n))} \vdash C}{\Gamma, A^P \vdash C}.$$

The first rule means that if u is a proof net for the sequent $\Gamma \vdash A$, there is a proof net u^P for the sequent $\Gamma^P \vdash A^P$. Note that in the second rule, we use multiplexing of rank n only, so that we get a restricted translation of SLL$_2$ into itself.

Theorem 3. *If the sequent $\Gamma \vdash A$ has a homogeneous proof of rank $P(n)$, then $\Gamma\langle P\rangle \vdash A\langle P\rangle$ has one of rank n.*

For instance, $A = !\alpha \multimap \alpha^9$ has a homogeneous proof of rank $9 = 3 \times 3$. Hence $A\langle XX\rangle = !!\alpha \multimap \alpha^9$ has a homogeneous proof of rank 3.



$\alpha \multimap \alpha$
$\alpha \multimap \alpha$
$\alpha \multimap \alpha$
α
α
α
α
$!(\alpha \multimap \alpha)$
$\alpha \multimap \alpha$
$!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$

Fig. 5. Net for number 3.

4. Natural numbers

The *type of natural numbers* is defined as follows:

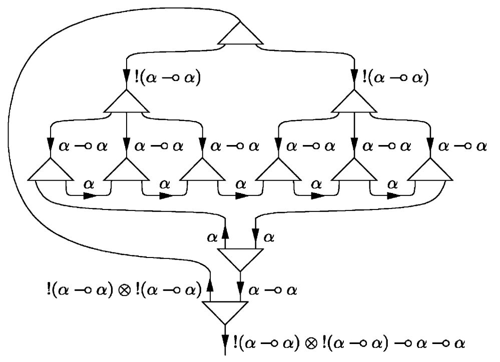
$$\mathbf{N} = \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha.$$

Its translation into second-order intuitionistic logic is indeed the type $\mathbf{N}_* = \forall\alpha.(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$ of *Church numerals* (see [5]). For instance, the net $\forall\alpha.u$ of type $\mathbf{N}$, where u is pictured in Fig. 5, corresponds to number 3: its translation into λ-calculus is indeed the Church numeral $\underline{3} = \lambda f.\lambda x.f(f(fx))$. Furthermore, it contains one multiplexor which is of rank 3. More generally, any irreducible net of type $\mathbf{N}$ corresponds to some natural number n: its translation is the Church numeral $\underline{n} = \lambda f.\lambda x.f(f(\cdots(fx)\cdots))$ and it contains exactly one multiplexor which is of rank n.

In fact, modulo commutativity of multiplexing, there is essentially one irreducible net of type $\mathbf{N}$ for each rank n. This means that, in the homogeneous case, $\mathbf{N}$ can be considered as a singleton. Note also that there is no generic proof of type $\mathbf{N}$. Using this, one shows that the sequent $\mathbf{N} \vdash \mathbf{N}^2$ has no generic proof, which means that *natural numbers are not duplicable*. However, the sequent $\mathbf{N} \vdash \mathbf{1}$ has a generic proof, which means that *natural numbers are erasable*. Details are given in the Appendix.

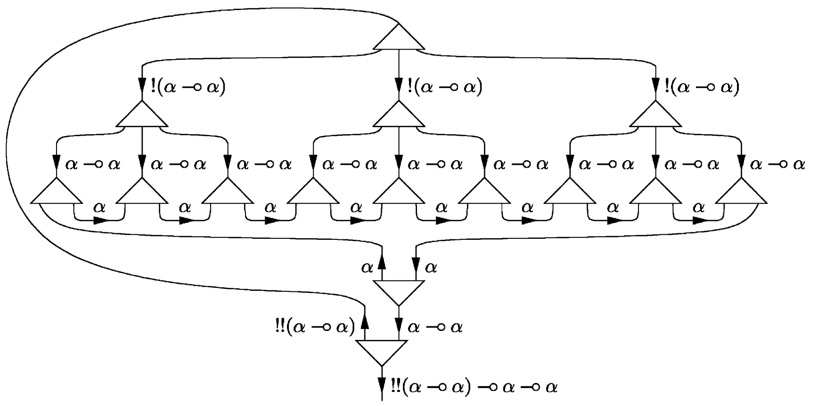
It is possible to build homogeneous proof nets of rank 3 corresponding to other natural numbers, but with different types. For instance:

- The net $\forall\alpha.u$ of type $\mathbf{N}\langle X + X\rangle = \forall\alpha.!(\alpha \multimap \alpha) \otimes !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$, where u is pictured in Fig. 6, corresponds to number $6 = 3 + 3$. The corresponding λ-term is indeed $\lambda(f,g).\lambda x.g(g(g(f(f(fx)))))$. It is not a Church numeral, but there is an obvious *coercion* from it to the Church numeral $\underline{6}$, which consists in merging the two variables f and g;
- The net $\forall\alpha.u$ of type $\mathbf{N}\langle XX\rangle = \forall\alpha.!!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$, where u is pictured in Fig. 7, corresponds to number $9 = 3 \times 3$. The corresponding λ-term is indeed the Church numeral $\underline{9}$.



$!(\alpha \multimap \alpha)$
$\alpha \multimap \alpha$
α
$!(\alpha \multimap \alpha) \otimes !(\alpha \multimap \alpha)$
$!(\alpha \multimap \alpha) \otimes !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$

Fig. 6. Net for number $6 = 3 + 3$.



$!(\alpha \multimap \alpha)$
$\alpha \multimap \alpha$
α
$!!(\alpha \multimap \alpha)$
$!!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$

Fig. 7. Net for number $9 = 3 \times 3$.

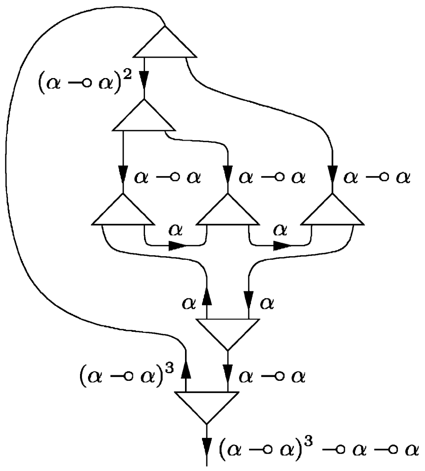
More generally, by applying Theorem 3 to the proof net of type **N** corresponding to $P(n)$, we get a proof net of rank n for each type $\mathbf{N}\langle P\rangle$, and any such net corresponds to $P(n)$. We shall see that this construction can be internalized in SLL_2.

Lemma 4. *If n is a natural number, there is a generic proof net of type* $\mathbf{N}\langle n\rangle$.

The type $\mathbf{N}\langle n\rangle = \forall\alpha.(\alpha \multimap \alpha)^n \multimap \alpha \multimap \alpha$ is a formula of IMALL_2. So, it suffices to apply the translation of SLL_2 into IMALL_2 to the proof net of type **N** corresponding to n. For instance, the generic net of type $\mathbf{N}\langle 3\rangle$ is $\forall\alpha.u$ where u is pictured in Fig. 8.

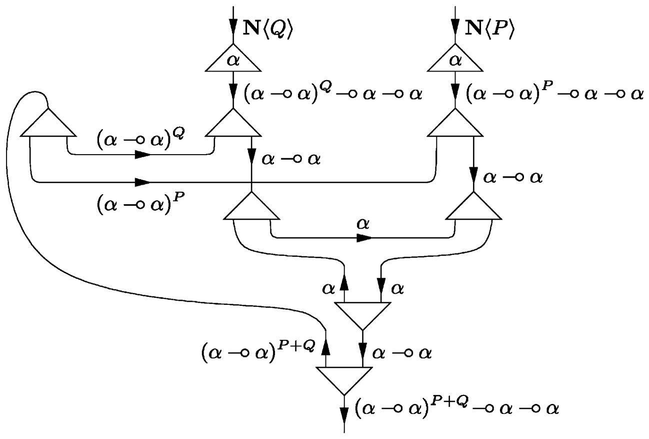
Lemma 5. *If P and Q are polynomial expressions, there are generic proofs for the following sequents*:

$$\mathbf{N}\langle P\rangle, \mathbf{N}\langle Q\rangle \vdash \mathbf{N}\langle P+Q\rangle, \quad \mathbf{N}\langle P\rangle, \mathbf{N}\langle Q\rangle \vdash \mathbf{N}\langle PQ\rangle.$$



$(\alpha \multimap \alpha)^2$
$\alpha \multimap \alpha$
$\alpha \multimap \alpha$
$\alpha \multimap \alpha$
α
α
α
α
$(\alpha \multimap \alpha)^3$
$\alpha \multimap \alpha$
$(\alpha \multimap \alpha)^3 \multimap \alpha \multimap \alpha$

Fig. 8. Generic net for number 3.



$\mathbf{N}\langle Q\rangle$
$\mathbf{N}\langle P\rangle$
α
α
$(\alpha \multimap \alpha)^Q \multimap \alpha \multimap \alpha$
$(\alpha \multimap \alpha)^P \multimap \alpha \multimap \alpha$
$(\alpha \multimap \alpha)^Q$
$\alpha \multimap \alpha$
$\alpha \multimap \alpha$
$(\alpha \multimap \alpha)^P$
α
α
α
$(\alpha \multimap \alpha)^{P+Q}$
$\alpha \multimap \alpha$
$(\alpha \multimap \alpha)^{P+Q} \multimap \alpha \multimap \alpha$

Fig. 9. Generic net for addition.

The corresponding nets are $\forall\alpha.u$ and $\forall\alpha.v$, where u and v are pictured in Figs. 9 and 10. By the above remark, they correspond necessarily to addition and multiplication, but this can also be checked directly:

- The λ-term corresponding to u is $\lambda(f,g).\lambda x.pf(qgx)$, where p and q are free variables of respective types $\mathbf{N}\langle P\rangle$ and $\mathbf{N}\langle Q\rangle$. After coercion, we get $\lambda f.\lambda x.pf(qfx)$, which is the usual term for addition.
- The λ-term corresponding to v is $\lambda f.p(q(\lambda g.\lambda x.f(gx))(\lambda y.y))$, which is a term for multiplication, but not the usual one $\lambda f.p(qf)$. The main difference is that q is applied to the functional $\lambda g.\lambda x.f(gx)$.

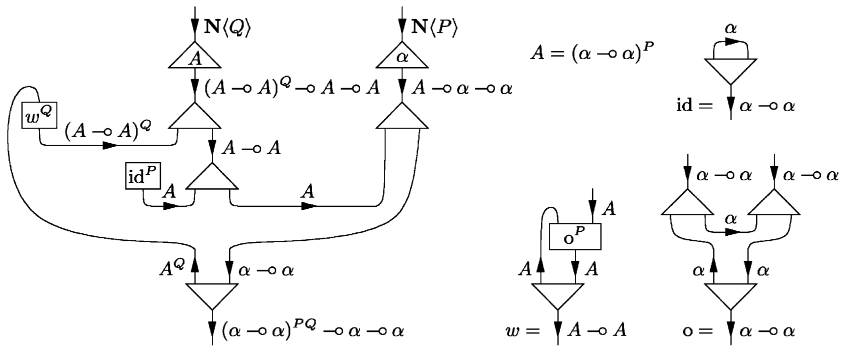


Fig. 10. Generic net for multiplication.

Since natural numbers are not duplicable in SLL_2, it will be necessary to allow several copies of $\mathbf{N}$ in our sequents. For that reason, we define the *pseudo-degree* δP of a polynomial expression P as follows:

$$\delta n = 0, \quad \delta X = 1, \quad \delta(P+Q) = \delta(PQ) = \delta P + \delta Q.$$

Note that if P is in Horner normal form, δP is just the degree of the associated polynomial.

Theorem 6. *If P is a polynomial expression, there is a generic proof for the sequent $\mathbf{N}^{(\delta P)} \vdash \mathbf{N}\langle P\rangle$.*

For instance, the following sequents have generic proofs:

$$\mathbf{N}^{(2)} \vdash \mathbf{N}\langle X+X\rangle, \quad \mathbf{N}^{(2)} \vdash \mathbf{N}\langle XX\rangle, \quad \mathbf{N} \vdash \mathbf{N}\langle 2X\rangle, \quad \mathbf{N} \vdash \mathbf{N}\langle X2\rangle.$$

The theorem is proved by induction on P, using Lemma 4 for $P = n$, the identity axiom for $P = X$, and Lemma 5 together with the cut rule for the remaining cases. Note the following points:

- Since $A^{X+X} = A^X \otimes A^X = A^{X2}$, we have $\mathbf{N}\langle X+X\rangle = \mathbf{N}\langle X2\rangle$, so that there is also a generic proof for the sequent $\mathbf{N} \vdash \mathbf{N}\langle X+X\rangle$. The reader is invited to find directly a cut-free proof for it.
- However, it seems that the exponent δP is necessary in the general case. For instance, we conjecture that there is no generic proof for the sequent $\mathbf{N} \vdash \mathbf{N}\langle XX\rangle$. Of course, it is always possible to get rid of this δP by using multiplexing: one gets a non-generic proof for the sequent $!\mathbf{N} \vdash \mathbf{N}\langle P\rangle$.

Finally, we consider the *type of pairs of natural numbers*, which must not be confused with $\mathbf{N}^2$:

$$\mathbf{P} = \forall\alpha.!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)^2.$$

For instance, the net $\forall\alpha.u$ of type $\mathbf{P}$, where u is pictured in Fig. 11, corresponds to the pair $(2,3)$. It is a homogeneous net of rank $2+3=5$, and the corresponding

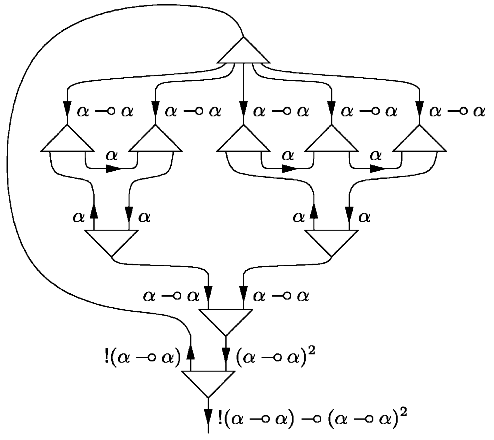


Fig. 11. Net for the pair (2, 3).

λ-term is $\lambda f.(\lambda x.f(f\,x), \lambda x.f(f(f\,x)))$. More generally, if the rank n is fixed, a net of type $\mathbf{P}$ corresponds to a pair (p,q) such that $p+q=n$. So, even in the homogeneous case, $\mathbf{P}$ is not a singleton. This type allows to represent the following map, which is a combination of predecessor and successor:

$$\psi(p,q) = (p-1, q+1) \quad \text{if} \quad p > 0, \quad \psi(0,n) = (0,n).$$

Lemma 7. *There is a generic proof for the sequent* $\mathbf{P} \vdash \mathbf{P}$ *which corresponds to the* ψ *map.*

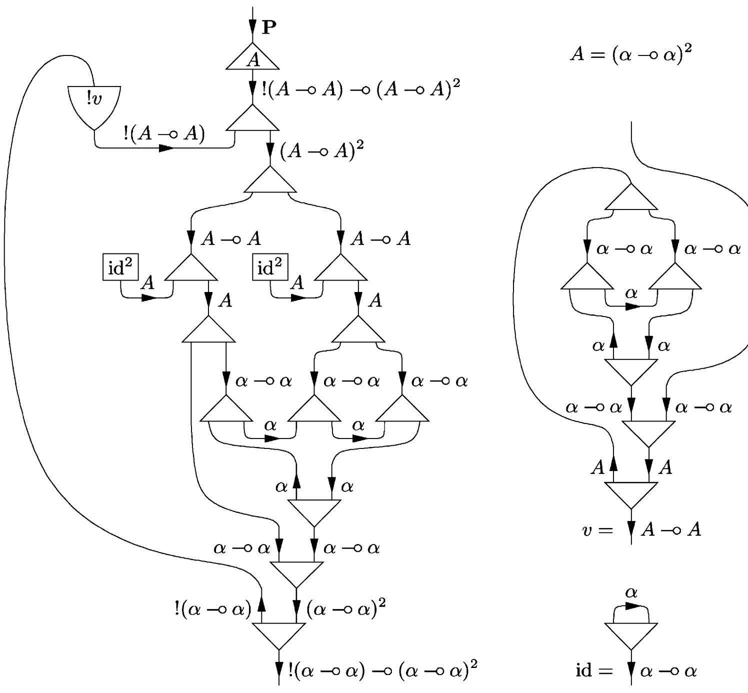
The corresponding net is $\forall\alpha.u$, where u is pictured in Fig. 12. The corresponding λ-term is a variation on Kleene's predecessor, which can be described informally as follows: Take f of type $\alpha \Rightarrow \alpha$ and consider the map F of type $(\alpha \Rightarrow \alpha)^2 \Rightarrow (\alpha \Rightarrow \alpha)^2$ defined by $F(g,h) = (h \circ g, f)$. Clearly, $F^n(\mathrm{id},\mathrm{id}) = (f^{n-1}, f)$. So, from $F^p(\mathrm{id},\mathrm{id}) = (f^{p-1}, f)$ and $F^q(\mathrm{id},\mathrm{id}) = (f^{q-1}, f)$, you can extract f^{p-1} and $f^{q+1} = f \circ f^{q-1} \circ f$.

5. Booleans, strings, and Turing machines

The *type of booleans* is defined as follows:

$$\mathbf{B} = \forall\alpha.(\alpha \& \alpha) \multimap \alpha.$$

The nets $\forall\alpha.u$ and $\forall\alpha.v$ of type $\mathbf{B}$, where u and v are pictured in Fig. 13, correspond to the boolean values 0 and 1. Furthermore, one can define a *conditional*: if u and v are two proof nets for the same sequent $\Gamma \vdash A$, there is a proof net w for the sequent $\Gamma, \mathbf{B} \vdash A$, which is pictured in Fig. 14, such that w reduces to u when it is connected to the net for 0, and to v when it is connected to the net for 1. An alternative definition for the type of booleans would be $\mathbf{B} = \mathbf{1} \oplus \mathbf{1}$, where the connective $\oplus$ is defined in



$\mathbf{P}$
A
$!(A \multimap A) \multimap (A \multimap A)^2$
$!v$
$!(A \multimap A)$
$(A \multimap A)^2$
$A \multimap A$
$A \multimap A$
id^2
A
A
$\alpha \multimap \alpha$
α
α
$\alpha \multimap \alpha$
$!(\alpha \multimap \alpha)$
$(\alpha \multimap \alpha)^2$
$!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)^2$
$A = (\alpha \multimap \alpha)^2$
$v =$
$A \multimap A$
$\mathrm{id} =$

Fig. 12. Generic net for the ψ map.

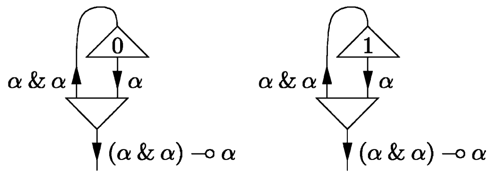


Fig. 13. Nets for the boolean values 0 and 1.

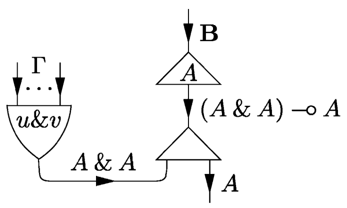


Fig. 14. Net for the conditional.

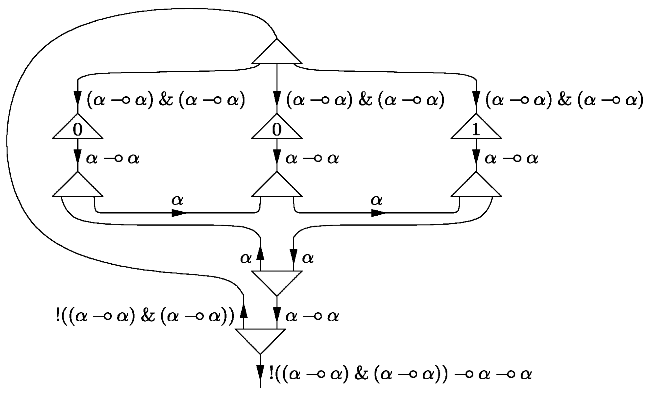


Fig. 15. Net for the string 001.

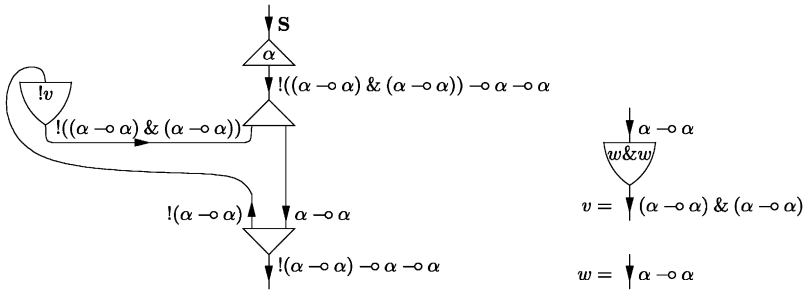


Fig. 16. Generic net for the length map.

Section 1. More generally, one defines a *finite type* $\mathbf{F}_n = \mathbf{1} \oplus \cdots \oplus \mathbf{1}$ (n times) for each natural number n.

The *type of boolean strings* is defined as follows:

$$\mathbf{S} = \forall\alpha.!((\alpha \multimap \alpha)\&(\alpha \multimap \alpha)) \multimap \alpha \multimap \alpha.$$

For instance, the net $\forall\alpha.u$ of type $\mathbf{S}$, where u is pictured in Fig. 15, corresponds to the string 001. It is a homogeneous net of rank 3. More generally, if the rank n is fixed, a net of type $\mathbf{S}$ corresponds to a string of length n. Furthermore, there is a generic net $\forall\alpha.u$ for the sequent $\mathbf{S} \vdash \mathbf{N}$, where u is pictured in Fig. 16, which corresponds to the *length* map.

A Turing machine with k states and 3 symbols (including *blank*) is represented by the following type:

$$\mathbf{M} = \forall\alpha.!((\alpha \multimap \alpha)\&(\alpha \multimap \alpha)) \multimap \mathbf{F}_k \otimes \mathbf{F}_3 \otimes (\alpha \multimap \alpha)^2.$$

Here, $\mathbf{F}_k$ stands for the current state and $\mathbf{F}_3$ for the current symbol which may be 0, 1, or *blank*. The remaining part of the type encodes the left and right sides of the tape, which can be seen as a pair of boolean strings. In fact, $\mathbf{M}$ generalizes $\mathbf{P}$, which corresponds to a simplified Turing machine with only one state and one symbol. In this encoding, it is essential that $\mathbf{F}_k$ and $\mathbf{F}_3$ occur *inside* the linear implication. We assume that our machine has a finite tape of fixed size (the *blank* symbol indicates the ends of the tape) and that the transition map is defined for all states, including the final ones.

Lemma 8. *For any Turing machine with k states and* 3 *symbols, there is a generic proof for the sequent* $\mathbf{M} \vdash \mathbf{M}$ *which corresponds to the transition map of this machine.*

We do not give the details here, but the idea is essentially the same as for Lemma 7, since the transition map generalizes ψ. Note the following points:

- Conditionals are necessary because the transition depends on the current state and symbol.
- The variable α must be substituted by $A = \mathbf{F}_3 \otimes (\alpha \multimap \alpha)^2$. This $\mathbf{F}_3$ is needed to produce the next current symbol, which comes from the left or from the right of the tape.

Similarly, it is possible to build:

- a generic proof net for the sequent $\mathbf{N} \vdash \mathbf{M}$ which transforms a natural number n into a machine with a tape of length n (filled with 0) and with the head at the beginning of the tape.
- a generic proof net for the sequent $\mathbf{M} \vdash \mathbf{M}$ which writes 0 (respectively 1) on the tape and moves the head to the right.
- a generic proof net for the sequent $\mathbf{M} \vdash \mathbf{B}$ which says if the machine is in an accepting state (assuming that final states are divided into accepting and non-accepting ones).

Theorem 9. *If a predicate on boolean strings is computable by a Turing machine in polynomial time* $P(n)$ *and in polynomial space* $Q(n)$, *there is a generic proof for the sequent* $\mathbf{S}^{(\deg P + \deg Q + 1)} \vdash \mathbf{B}$ *which corresponds to this predicate.*

We assume that P and Q are in Horner normal form. By Lemma 8 and Theorem 3, there is a generic proof net for the sequent $\mathbf{M}\langle Q\rangle \vdash \mathbf{M}\langle Q\rangle$ which corresponds to the transition map of the machine (with a tape of size $Q(n)$), from which we get a generic proof net of type $(\mathbf{M}\langle Q\rangle \multimap \mathbf{M}\langle Q\rangle)^P$. So we have a generic proof net for the sequent $\mathbf{N}\langle P\rangle, \mathbf{M}\langle Q\rangle \vdash \mathbf{M}\langle Q\rangle$ which corresponds to the full computation of the machine (with $P(n)$ transitions). By using the net which computes the length of a boolean string and the one which generates a machine, together with Theorems 3 and 6, we also get generic proof nets for the sequents $\mathbf{S}^{(\deg P)} \vdash \mathbf{N}\langle P\rangle$ and $\mathbf{S}^{(\deg Q)} \vdash \mathbf{M}\langle Q\rangle$. Finally, we have a generic proof net for the sequent $\mathbf{S}, \mathbf{M}\langle Q\rangle \vdash \mathbf{M}\langle Q\rangle$ which writes a string of size n on a tape of size $Q(n)$ (assuming $n \leqslant Q(n)$) and one for the sequent $\mathbf{M}\langle Q\rangle \vdash \mathbf{B}$ which says if the machine is in an accepting state. By combining all those nets, we get a generic proof net for the sequent $\mathbf{S}^{(\deg P + \deg Q + 1)} \vdash \mathbf{B}$ which corresponds to the predicate.

6. Conclusion

We have seen how data such as boolean strings are represented by homogeneous nets and programs by generic nets. By Theorem 2, any generic proof net for the sequent $\mathbf{S}^{(n)} \vdash \mathbf{B}$ defines a polynomial time algorithm that takes a boolean string as input and returns a boolean value as output. Conversely, by Theorem 9, any such polynomial time algorithm is represented by a generic proof net. In other words, SLL_2 is complete for polynomial time computation. Here are some possible variations on SLL_2:

- The classical version of SLL_2 would work as well. We chose the intuitionistic one because the sequents are easier to read and to relate with λ-calculus.
- As discussed at the end of Section 2, fixpoints of types can be introduced without loosing the nice properties of the system. In that case, the proof of Theorem 9 is simpler, since a Turing machine with k states and 2 symbols can be represented by a net of type $\mathbf{F}_k \otimes \mathbf{F}_3 \otimes T^2$ where T is the fixpoint $T = \mathbf{1} \oplus (\mathbf{B} \otimes T)$.
- We conjecture that *multiplicative* SLL_2, that is SLL_2 without additives, is also complete for polynomial time computation. In that case, an alternative encoding is used: for instance, booleans are represented by $\forall\alpha.\alpha^2 \multimap \alpha^2$.
- It may be convenient to introduce several modalities $!_X$, $!_Y$, $!_Z, \ldots$, indexed by *size variables* $X, Y, Z, \ldots$ and quantification over size variables with suitable rules.
- It may also be interesting to introduce a hierarchy of modalities $!_0$, $!_1$, $!_2$, … with a hierarchical version of *digging*:

  $$\frac{\Gamma, !_n \cdots !_n A \vdash C}{\Gamma, !_{n+1} A \vdash C}.$$

  In that case, computation is no more polynomial: one gets a system for elementary recursive computation, as in *Elementary linear logic*.
- Finally, one could investigate restrictions on *interaction systems*, possibly variants of *interaction combinators* (see [8]), with the same computational properties as SLL_2.

Appendix

Fact 10. *There is no generic proof for the sequent* $\vdash \mathbf{N}$.

Indeed, a cut-free proof of this sequent ends necessarily as follows:

$$\dfrac{\dfrac{\dfrac{!(\alpha \multimap \alpha), \alpha \vdash \alpha}{!(\alpha \multimap \alpha) \vdash \alpha \multimap \alpha}}{\vdash !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha}}{\vdash \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha.}$$

Without multiplexing, there is no way to prove the sequent $!(\alpha \multimap \alpha), \alpha \vdash \alpha$.

Fact 11. *There is no generic proof for the sequent* $\mathbf{N} \vdash \mathbf{N} \otimes \mathbf{N}$ (*natural numbers are not duplicable*).

Indeed, by the previous result, a generic cut-free proof of this sequent ends necessarily as follows:

$$\dfrac{\dfrac{\vdash !(A \multimap A) \qquad \dfrac{\vdash A \quad A \vdash \mathbf{N} \otimes \mathbf{N}}{A \multimap A \vdash \mathbf{N} \otimes \mathbf{N}}}{!(A \multimap A) \multimap A \multimap A \vdash \mathbf{N} \otimes \mathbf{N}}}{\forall \alpha . !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha \vdash \mathbf{N} \otimes \mathbf{N}}.$$

Here, the formula A is unknown, but if there is a generic proof for $\vdash A$ and one for $A \vdash \mathbf{N} \otimes \mathbf{N}$, there is also one for $\vdash \mathbf{N} \otimes \mathbf{N}$, which contradicts the previous result.

Fact 12. *There is a generic proof for the sequent* $\mathbf{N} \vdash \mathbf{1}$ (*natural numbers are erasable*).

The proof is the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{1} \vdash \mathbf{1}}}{\vdash \mathbf{1} \multimap \mathbf{1}}}{\vdash !(\mathbf{1} \multimap \mathbf{1})} \qquad \dfrac{\dfrac{}{\vdash \mathbf{1}} \quad \dfrac{}{\mathbf{1} \vdash \mathbf{1}}}{\mathbf{1} \multimap \mathbf{1} \vdash \mathbf{1}}}{!(\mathbf{1} \multimap \mathbf{1}) \multimap \mathbf{1} \multimap \mathbf{1} \vdash \mathbf{1}}}{\forall \alpha . !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha + \mathbf{1}}.$$

References

[1] A. Asperti, Light affine logic, Thirteenth Annual IEEE Symp. on Logic in Computer Science, IEEE Computer Society Press, Silver Spring, MD, 1991, pp. 300–308.
[2] V. Danos, L. Regnier, The structure of multiplicatives, Arch. Math. Logic 28 (1989) 181–203.
[3] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1987) 1–102.
[4] J.-Y. Girard, Light linear logic, Inform. and Comput. 143 (1998) 175–204.
[5] J.-Y. Girard, Y. Lafont, P. Taylor, Proofs and Types, in: Cambridge Tracts in Theoretical Computer Science, Vol. 7, Cambridge University Press, Cambridge, 1989.
[6] J.-Y. Girard, A. Scedrov, P. Scott, Bounded linear logic: a modular approach to polynomial time computability, Theoret. Comput. Sci. 97 (1992) 1–66.
[7] Y. Lafont, From proof-nets to interaction nets, in: J.-Y. Girard, Y. Lafont & L. Regnier (Eds.), Advances in Linear Logic, London Mathematical Society Lecture Note Series, Vol. 222, Cambridge University Press, Cambridge, 1995, pp. 225–247.
[8] Y. Lafont, Interaction combinators, Inform. Comput. 137 (1) (1997) 69–101.
[9] D. Leivant, A foundational delineation of computational feasibility, Sixth Annual IEEE Symp. on Logic in Computer Science, IEEE Computer Society Press, Silver Spring, MD, 1991, pp. 69–101.
[10] L. Roversi, A *P*-time completeness proof for light logics, Ninth Annual Conf. of the EACSL (CSL'99), Lecture Notes in Computer Science, Vol. 1683, Springer, Berlin, 1999, pp. 469–483.