From light logics to type assignments: a case study

MARCO GABOARDI, *Dipartimento di Informatica, Università degli Studi di Torino. Corso Svizzera 185, 10149 Torino, Italy.*
E-mail: gaboardi@di.unito.it

SIMONA RONCHI DELLA ROCCA, *Dipartimento di Informatica, Università degli Studi di Torino. Corso Svizzera 185, 10149 Torino, Italy.*
E-mail: ronchi@di.unito.it

Abstract

Using Soft Linear Logic (SLL) as case study, we analyze a method for transforming a light logic into a type assignment system for the λ-calculus, inheriting the complexity properties of the logics. Namely the typing assures the strong normalization in a number of steps polynomial in the size of the term, and moreover all polynomial functions can be computed by λ-terms that can be typed in the system. The proposed method is general enough to be used also for other light logics.

Keywords: type assignment, implicit computational complexity, lambda calculus, polynomial time

1 Introduction

The light logics, Light Linear Logic (LLL) [15], Soft Linear Logic (SLL) [17] and Elementary Linear Logic (ELL) [8], were introduced as logical counterparts of some computational complexity classes. Proofs of LLL and SLL characterize polynomial time computations, while ELL characterizes elementary time computations. The characterization is based on the fact that proofs of these logics normalize in a number of cut-elimination steps which is either polynomial (in case of LLL and SLL) or elementary (in case of ELL) in their size, if their depth is fixed, and moreover they can encode every function with the given complexity.

From a computer science perspective, light logics can be used for the design of programming languages with a given computational bound. This could be done in a straightforward way by a complete decoration of the logical proofs, but, due to the presence of modalities, the resulting languages could have a very complex syntactical structure, and they cannot be reasonably proposed for programming (an example of complete decoration of SLL is in [3]). A different approach is to fix as starting points:
1. The use of λ-calculus as an abstract paradigm of programming languages.
2. The use of types to characterize program properties.
In this line, the aim becomes the design of a type assignment system for λ-calculus, where types are formulae of a light logic, in such a way that the logical properties are inherited by the well typed terms. Then types can be used for checking, besides the usual notion of correctness, also the complexity properties.

Some results have already been obtained in this line. Two different proposals for a polynomial λ-calculus have been designed by Baillot and Terui [4] and Gaboardi and Ronchi Della

Vol. 17 No. 5,
doi:10.1093/jigpal/jzp019

Rocca [12], starting respectively from LAL, a simplified version of LLL defined in [1, 2], and from SLL. A proposal for an elementary λ-calculus has been given in Coppola, Dal Lago and Ronchi Della Rocca [7].

In all cases the starting point is the Curry-Howard isomorphism, connecting proofs with programs, formulae with types and proof normalization with β-reduction. But the modal aspect of the light logics breaks the last connection, since the cut elimination does not correspond naturally to β-reduction, and so a proof decoration following the standard path gives rise to a type assignment where the subject reduction is lost, and the complexity properties are no more preserved. In all cases cited before, the problem has been solved not making directly a decoration of the logical proofs by λ-terms, but by designing a type assignment system in some sense "inspired" to the principles of the logics, in such a way to obtain the same complexity bound, and moreover the complexity bound for the language is not directly inherited from the logics, but it has been proved again, and in general these proofs are quite involved. An exception is [5], where there is an alternative proof of the complexity bound for the language in [4], that is derived from the bound of the logic in an indirect way.

The aim of this paper is to deeply explore and further enhance this approach. We will use, as case study, SLL, but we stress that the same kind of procedure can be adapted to all the other light logics. The idea is to modify the logic, while preserving the complexity properties, until a natural deduction version of it is reached, with the property that the standard decoration of it by λ-terms is the type assignment system with the desired properties. The transformation is given through the following path. First of all, a deep analysis of SLL allows us to replace the cut rule by three different rules, according to the syntactical shape of their premises. Just one of these new cuts does not correspond correctly to a β-reduction. Then we design a new logic, by restricting SLL in such a way that a cut with the "bad" behavior never appears. This new logic, which we call Essential Soft Linear Logic (ESLL), preserves the good properties of SLL with respect to the complexity bound. Then we design a natural deduction version of ESLL, namely NESLL, enjoying the desired properties too. A decoration of NESLL proofs by λ-terms, in the standard way, gives rise to the desired type assignment system, named STA. So we obtain for free a polynomial bound for STA, inherited from the logic. This bound is obviously expressed in function of some measures of the typing proof, so some further work is necessary, in order to relate it to the size of the term.

In order to do so, some intermediate results are needed, which are interesting by themselves, in our opinion. First of all, the properties of SLL have been proved by Lafont using proof-nets, which are graph representation of equivalence classes of proofs. Since we want to obtain the type assignment through a decoration of the logic, we need in particular to rephrase the strong normalization and its polynomial bound directly on the proofs, the technical difficulty being that in this case we need to take into account also the commutation steps, which are not needed in the normalization of proof-nets. As far as we know, this is the only normalization proof for a light logics not based on proof-nets. Moreover, all the type assignment systems based on light logics use an affine version of them, being it easier for the completeness proofs, while we use SLL in the original, not affine, version.

The completeness with respect to the polynomial computations is proved directly for STA. The completeness we are talking about is a functional completeness, in the sense that we prove (through a simulation of polynomial time Turing Machines) that all polynomial

functions can be computed by terms typable in STA. For the completeness, we are not interested in inheriting the property from the one of SLL, since we will prove a stronger result. In fact, STA is complete for FPTIME, while Lafont proved SLL complete for PTIME only.

A type assignment for λ-calculus derived from an affine version of ESLL has been already presented in [12] and [9]. This system has been also used in [11] and [10].

The paper is organized as follows. In Section 2, SLL is introduced, its strong normalization in polynomial time is proved and its PTIME completeness is recalled. In Section 3, a deep analysis is given on the mismatch between cut elimination and β-reduction for light logics, and the methods used in the literature for solving this problem are examined. Section 4 contains ESLL and its natural deduction version, NESLL. In Section 5 the type assignment system STA is presented, and its properties are proved.

2 Soft Linear Logic

In this section we recall the fragment of Soft Linear Logic (SLL) restricted to the connectives $\multimap, \forall$ and the modality !. For this fragment, we give a proof of the polynomial soundness and we sketch the polynomial completeness.

Definition 2.1

(i) The set of formulae of SLL is defined by the following grammar:

$$A, B, C ::= \alpha \mid A \multimap A \mid \forall \alpha. A \mid !A$$

where α ranges over a countable set of variables.

(ii) SLL contexts are multisets of SLL formulae. Contexts are ranged over by Γ, Δ. $\mathrm{FV}(\Gamma)$ denotes the set of free variables occurring in the formulae of Γ while $|\Gamma|$ denotes the cardinality of Γ, i.e. the number of occurrences of formulae in Γ. The notation $!\Gamma$ is a short for $\{!A \mid A \in \Gamma\}$.

(iii) The set of SLL rules prove judgements of the shape:

$$\Gamma \vdash A$$

where Γ is a context and A is a formula. The rules are given in Table 1, where $A^{(n)}$ denotes $\underbrace{A, ..., A}_{n}$. As usual $\vdash A$ is a short for $\emptyset \vdash A$.

(iv) Derivations are denoted by $\Pi, \Sigma, \Phi, \Psi, \Theta$. $\Pi \rhd \Gamma \vdash A$ denotes a derivation Π with conclusion $\Gamma \vdash A$.

SLL is a restriction of Girard's Linear Logic (LL) [14], obtained in two steps. First, by replacing the rules of LL dealing with the modality !:

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\ (!R) \qquad \frac{\Gamma, B \vdash A}{\Gamma, !B \vdash A}\ (!L)$$

and the structural rules of weakening and contraction:

$$\frac{\Gamma \vdash A}{\Gamma, !B \vdash A}\ (W) \qquad \frac{\Gamma, !B, !B \vdash A}{\Gamma, !B \vdash A}\ (C)$$

TABLE 1. Soft Linear Logic

$$\frac{}{A\vdash A}\ (Ax) \qquad \frac{\Gamma\vdash A \quad B,\Delta\vdash C}{\Gamma,A\multimap B,\Delta\vdash C}\ (\multimap L) \qquad \frac{\Gamma\vdash A \quad \alpha\notin \mathrm{FV}(\Gamma)}{\Gamma\vdash \forall\alpha.A}\ (\forall R)$$

$$\frac{\Gamma\vdash A \quad \Delta,A\vdash B}{\Gamma,\Delta\vdash B}\ (cut) \qquad \frac{\Gamma,A\vdash B}{\Gamma\vdash A\multimap B}\ (\multimap R) \qquad \frac{\Gamma,A[B/\alpha]\vdash C}{\Gamma,\forall\alpha.A\vdash C}\ (\forall L)$$

$$\frac{\Gamma\vdash A}{!\Gamma\vdash !A}\ (sp) \qquad \frac{\Gamma,A^{(n)}\vdash B \quad n\geq 0}{\Gamma,!A\vdash B}\ (m)$$

by the three rules of multiplexor, soft promotion and digging:

$$\frac{\Gamma,A^{(n)}\vdash B}{\Gamma,!A\vdash B}\ (m) \qquad \frac{\Gamma\vdash A}{!\Gamma\vdash !A}\ (sp) \qquad \frac{\Gamma,!!B\vdash A}{\Gamma,!B\vdash A}\ (digging)$$

Note that the (m) rule is parametric in the number n, which is its *rank*. The resulting system is equivalent to LL. The weakening rule is a particular case of multiplexor, with $n=0$. The contraction rule can be obtained by (m) followed by $(digging)$.
The second step is to erase the rule $(digging)$. The result is that it doesn't hold anymore the linear correspondence:

$$!A\multimap !!A.$$

As a consequence, the modality ! can be used for counting the number of duplications of (sub)proofs.

2.1 Polynomial time soundness

In this subsection we prove that Soft Linear Logic is correct for polynomial time computations. This result has been already proved by Lafont, in [17], using the proof-nets representation of proofs. Here we rephrase his proof using directly the sequent calculus formulation, in order to reuse it in the construction of a related type assignment system for λ-calculus. First we need to define some measures on SLL proofs. Note that the key measure, the weight of a proof, is different from the notion of weight used by Lafont: the change is necessary for dealing with the new setting.

DEFINITION 2.2

- The *size* of a proof Π, denoted by $|\Pi|$, is the number of rules applications in Π.
- The *rank* of a proof Π, denoted by $\mathtt{rk}(\Pi)$, is the maximal rank of multiplexor rules in Π.
- The *degree* of a proof Π, denoted by $\mathtt{d}(\Pi)$, is the maximal nesting of applications of the (sp) rule in Π, i.e., the maximal number of applications of the (sp) rule in a path connecting the conclusion and one axiom of Π.
- Let r be a natural number. The *weight* $\mathtt{W}(\Pi,r)$ of Π with respect to r is defined inductively as follows:

– If the last applied rule is (Ax) then $\mathtt{W}(\Pi, r) = 1$.
– If the last applied rule is:

$$\frac{\Sigma \rhd \Gamma \vdash A}{!\Gamma \vdash !A}\ (sp)$$

then $\mathtt{W}(\Pi, r) = (r+1) \times (\mathtt{W}(\Sigma, r)+1)$.
– In any other case (also the cut case!) $\mathtt{W}(\Pi, r)$ is the sum of the weights of the premises with respect to r plus 1 .

The following lemma follows directly from the definition of measures.

Lemma 2.3
For every SLL proof Π:

(i) $\mathtt{W}(\Pi, 0) = |\Pi|$
(ii) $\mathtt{W}(\Pi, \mathtt{rk}(\Pi)) \leq \mathtt{W}(\Pi, 0) \times (\mathtt{rk}(\Pi)+1)^{\mathtt{d}(\Pi)} \leq |\Pi|^{\mathtt{d}(\Pi)+1}$

Now we will prove that the cut elimination property holds for SLL. In order to do this, we need to define the cut elimination steps.

Definition 2.4
(i) A *symmetric* step is a proof rewriting rule with one of the following shapes:

Case (R)-(Ax), where (R) is any rule:

$$\frac{\Sigma \rhd \dfrac{}{\Gamma \vdash A}\ (R) \qquad \dfrac{}{A \vdash A}\ (Ax)}{\Gamma \vdash A}\ (cut) \qquad \text{rewrites to} \qquad \Sigma \rhd \Gamma \vdash A$$

Case (sp)-(m):

$$\frac{\dfrac{\Sigma \rhd \Gamma \vdash A}{!\Gamma \vdash !A}\ (sp) \qquad \dfrac{\Theta \rhd A^{(n)}, \Delta \vdash B}{!A, \Delta \vdash B}\ (m)}{!\Gamma, \Delta \vdash B}\ (cut)$$

in case $n = 0$ rewrites to:

$$\frac{\Theta \rhd \Delta \vdash B}{!\Gamma, \Delta \vdash B}\ (m)$$

otherwise it rewrites to:

$$\frac{\Sigma \rhd \Gamma \vdash A \qquad \dfrac{\Sigma \rhd \Gamma \vdash A \quad \Theta \rhd \Delta, A^{(n)} \vdash B}{\Gamma^{(n-1)}, A, \Delta \vdash B}\ (cut)}{\dfrac{\Gamma^{(n)}, \Delta \vdash B}{!\Gamma, \Delta \vdash B}\ (m)}\ (cut)$$

where the double line denotes a number ≥ 0 of applications of the rule.

Case (sp)-(sp):

$$\dfrac{\dfrac{\dfrac{\Sigma \rhd \Gamma \vdash A}{!\Gamma \vdash !A}\,(sp) \quad \dfrac{\Theta \rhd A, \Delta \vdash B}{!A, !\Delta \vdash !B}\,(sp)}{!\Gamma, !\Delta \vdash !B}\,(cut)}{} \text{ rewrites to } \dfrac{\dfrac{\Sigma \rhd \Gamma \vdash A \quad \Theta \rhd A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\,(cut)}{!\Gamma, !\Delta \vdash !B}\,(sp)$$

Case $(\multimap R)$-$(\multimap L)$:

$$\dfrac{\dfrac{\Sigma \rhd \Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\,(\multimap R) \quad \dfrac{\Theta_1 \rhd \Delta_1 \vdash A \quad \Theta_2 \rhd B, \Delta_2 \vdash C}{\Delta_1, A \multimap B, \Delta_2 \vdash C}\,(\multimap L)}{\Gamma, \Delta_1, \Delta_2 \vdash C}\,(cut)$$

rewrites to:

$$\dfrac{\dfrac{\Theta_1 \rhd \Delta_1 \vdash A \quad \Sigma \rhd \Gamma, A \vdash B}{\Gamma, \Delta_1 \vdash B}\,(cut) \quad \Theta_2 \rhd B, \Delta_2 \vdash C}{\Gamma, \Delta_1, \Delta_2 \vdash C}\,(cut)$$

Case $(\forall R)$-$(\forall L)$:

$$\dfrac{\dfrac{\Sigma \rhd \Gamma \vdash A}{\Gamma \vdash \forall \alpha . A}\,(\forall R) \quad \dfrac{\Theta \rhd A[C/\alpha], \Delta \vdash B}{\forall \alpha . A, \Delta \vdash B}\,(\forall L)}{\Gamma, \Delta \vdash B}\,(cut)$$

rewrites to:

$$\dfrac{\Sigma[C/\alpha] \rhd \Gamma \vdash A[C/\alpha] \quad \Theta \rhd \Delta, A[C/\alpha] \vdash B}{\Gamma, \Delta \vdash B}\,(cut)$$

where $\Sigma[C/\alpha]$ denotes a derivation obtained from Σ by replacing every occurrence of α by the formula C.

(ii) A *commutative* step is a proof rewriting rule defined as follows. Let Π be:

$$\dfrac{\dfrac{}{\Gamma \vdash A}\,(R) \quad \dfrac{}{\Delta, A \vdash B}\,(R')}{\Gamma, \Delta \vdash B}\,(cut)$$

where (R) and (R') are rules such that at least one of them is different from (cut) and is not building the formula A (so we will say that A is passive in it).
Let A be passive in (R), which is not (cut). Then

$$\dfrac{\dfrac{\Sigma \rhd \Gamma' \vdash A \quad \Theta}{\Gamma \vdash A}\,(R) \quad \dfrac{}{\Delta, A \vdash B}\,(R')}{\Gamma, \Delta \vdash B}\,(cut)$$

rewrites to:

$$\dfrac{\Sigma \rhd \Gamma' \vdash A \quad \dfrac{}{\Delta, A \vdash B}\,(R')}{\dfrac{\Gamma', \Delta \vdash B}{\Gamma, \Delta \vdash B}\,\Theta}\,(cut)\ (R)$$

where, of course, Θ could not occur.
Let A be passive in (R'), which is not (cut). Then

$$\dfrac{\dfrac{}{\Gamma \vdash A}\,(R) \quad \dfrac{\Sigma \rhd \Delta', A \vdash C \quad \Theta}{\Delta, A \vdash B}\,(R')}{\Gamma, \Delta \vdash B}\,(cut)$$

rewrites to:

$$\dfrac{\Gamma \vdash A \quad \dfrac{\Sigma \rhd \Delta', A \vdash C}{\Gamma, \Delta' \vdash C}\,(cut)}{\Gamma, \Delta \vdash B}\,\Theta\ (R')$$

where, of course, Θ could not occur.

Let a proof be *normal* if it does not contain applications of the rule (cut).
The following result is the key point for obtaining the polynomial bound.

Lemma 2.5
Let Π rewrite to Π' in one cut elimination step, and let $r \geq \mathtt{rk}(\Pi)$. Then

(i) If the step is symmetric, then $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$;
(ii) If the step is commutative, then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Pi', r)$;

Proof.

(i) The proof is easy, by considering all the possible cases. We just show the most difficult one, the case (sp)-(m). Let Π be:

$$\dfrac{\dfrac{\Sigma \rhd \Gamma \vdash A}{!\Gamma \vdash !A}\,(sp) \quad \dfrac{\Theta \rhd A^{(n)}, \Delta \vdash B}{!A, \Delta \vdash B}\,(m)}{!\Gamma, \Delta \vdash B}\,(cut)$$

and let $n > 0$. So Π' is:

$$\dfrac{\Sigma \rhd \Gamma \vdash A \quad \dfrac{\dfrac{\Theta \rhd \Delta, A^{(n)} \vdash B}{\Gamma^{(n-1)}, A, \Delta \vdash B}\,(cut)}{\dfrac{\Gamma^{(n)}, \Delta \vdash B}{!\Gamma, \Delta \vdash B}\,(m)}\,(cut)}{}$$

clearly we have $\mathtt{W}(\Pi, r) = (r+1) \times (\mathtt{W}(\Sigma, r)+1) + (\mathtt{W}(\Theta, r)+1) + 1$ while $\mathtt{W}(\Pi', r) = n \times (\mathtt{W}(\Sigma, r)+1) + \mathtt{W}(\Theta, r) + |\Gamma|$, so for $r \geq \mathtt{rk}(\Pi) \geq n$ since $\mathtt{W}(\Sigma, r) \geq |\Gamma|$ we have:

$$\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$$

(ii) Immediate, by the definition of commutative step. ∎

THEOREM 2.6
Let Π be a SLL proof. Then Π rewrites to a normal proof Π' by a number of cut elimination steps bounded by $2\times|\Pi|^{3\times(\mathrm{d}(\Pi)+1)}$.

PROOF. The proof is by induction on the pair $<\mathtt{W}(\Pi,\mathtt{rk}(\Pi)),h(\Pi)>$, ordered by lexicographic order, where $h(\Pi)$ is the sum of the heights of all the subproofs of Π whose root is an application of the (cut) rule.
Consider the case where the cut elimination step is a symmetric step. Then $\mathtt{W}(\Pi,\mathtt{rk}(\Pi))>\mathtt{W}(\Pi',\mathtt{rk}(\Pi'))$ by Lemma 2.5.(i). Applications of commutative steps do not change the weight (by Lemma 2.5.(ii)). Nevertheless, by inspecting the Definition 2.4.(ii), it is easy to verify that they decrease the measure $h(\Pi)$. So the normalization procedure always stops, and the proof can be normalized.
Now consider the number n of cut reduction steps to normal form. Π clearly normalizes in a number n_s of symmetric steps bounded by $\mathtt{W}(\Pi,\mathtt{rk}(\Pi))$. The number of commutative steps n_c is $\Sigma_{0\le i\le n_s} n_c^i$, where n_c^i is the number of commutative steps performed after the i-th symmetric step. Note that every n_c^i is bound by the square of the maximum size of the proof during the rewriting, which, by Lemma 2.3.(i), can be bounded by $\mathtt{W}(\Pi,\mathtt{rk}(\Pi))$. Hence we have:

$$\begin{array}{rcl} n & \le & n_s+n_c\le n_s+n_s\times\max(n_c^i)\le n_s\times(\max(n_c^i)+1)\\ & \le & \mathtt{W}(\Pi,\mathtt{rk}(\Pi)\times(\max(n_c^i)+1)\le\mathtt{W}(\Pi,\mathtt{rk}(\Pi)\times(\mathtt{W}(\Pi,\mathtt{rk}(\Pi)^2+1)\\ & \le & 2\times\mathtt{W}(\Pi,\mathtt{rk}(\Pi))^3 \end{array}$$

and by Lemma 2.3.(ii) we can conclude:

$$n\le 2\times|\Pi|^{3\times(\mathrm{d}(\Pi)+1)}$$

∎

Since every proof has a fixed depth, the bound is polynomial.

2.2 PTIME Completeness

SLL is complete for PTIME. We will not give here the details of the proof given in [17] since the completeness proof for the type assignment we will design does not use them and moreover we will prove a stronger result.
Here we give some hints just to give evidence of the difference between our codings and that used by Lafont. First of all, Lafont proof uses data type defined using the connectives $\otimes$ and $\&$. Instead we will define data type considering these connectives in their derivate form in the implicative and exponential second order fragment of SLL.
Moreover, he introduces a programming discipline for Soft Linear Logic which consists in representing programs by generic proofs, i.e. proofs that do not contain occurrences of (m) rule, and data by homogeneous proofs, proofs where all the occurrences of rule (m) have the same rank, while we do not need such a distinction.
Let the symbol $\doteq$ denote definitional equivalence. If $\mathbf{N}\doteq\forall\alpha.!(\alpha\multimap\alpha)\multimap\alpha\multimap\alpha$ then natural numbers are definable by homogeneous proofs deriving the sequent:

$$\vdash\mathbf{N}$$

Polynomials in one variable are sufficient to our scope. For technical reasons instead of polynomials Lafont has introduced a notion of *polynomial expressions*, terms built from natural numbers and a variable X, using addition and multiplication. The following notation will be useful in the sequel:

$$A^n \doteq \overbrace{A \otimes \cdots \otimes A}^{n\text{-times}} \qquad A^X \doteq !A \qquad A^{P+Q} \doteq A^P \otimes A^Q \qquad A^{PQ} \doteq (A^P)^Q$$

Polynomial expressions are in one to one correspondence with polynomials in *Horner normal form*, i.e. of the form $a_0 + X(a_1 + X(\cdots(a_{n-1} + Xa_n)\cdots)$. If P is a polynomial expression we denote by $\delta(P)$ its degree. The formula representing natural numbers can be extended to polynomial expressions, i.e. $\mathbf{N}\langle P\rangle \doteq \forall\alpha.(\alpha \multimap \alpha)^P \multimap \alpha \multimap \alpha$. Note that $\mathbf{N}\langle X\rangle \equiv \mathbf{N}$.
The following theorem assures that all polynomial expression can be represented in SLL.

Theorem 2.7
If P is a polynomial expression, there is a generic proof for the sequent:

$$\mathbf{N}^{(\delta(P))} \vdash \mathbf{N}\langle P\rangle$$

The above theorem corresponds to a polynomial iteration principle for Soft Linear Logic. Booleans and boolean strings are definable in SLL by proofs of the following formulae:

$$\mathbf{B} \doteq \forall\alpha.(\alpha \& \alpha) \multimap \alpha \qquad \mathbf{S} \doteq \forall\alpha.!((\alpha \multimap \alpha) \& (\alpha \multimap \alpha)) \multimap \alpha \multimap \alpha$$

We refer to [17] for the definition of Turing machine configurations. Here we only recall the PTIME completeness theorem for SLL.

Theorem 2.8
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:

$$\mathbf{S}^{(\delta(P)+\delta(Q)+1)} \vdash \mathbf{B}$$

which corresponds to this predicate.

3 SLL and λ-calculus

There is a standard way to decorate the proofs of a logic in sequent calculus style in order to obtain a type assignment system for the λ-calculus [16][20]. By applying it to SLL, we obtain the type assignment system defined in [18] as a technical tool for studying the expressive power of SLL. This decoration, that we name SLL_λ, is presented in Table 2, where, by an abuse of notation, we use Γ, Δ to denote SLL_λ contexts i.e., variable assignments of the shape $\mathtt{x}:A$. Moreover, $\mathrm{dom}(\Gamma)$ denotes the set $\{\mathtt{x} \mid \mathtt{x}:A \in \Gamma\}$.
In SLL_λ the subject reduction property fails. Let us show it by an example.

Example 3.1
Consider the term $\mathtt{M} \equiv \mathtt{y}((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})$ and let $C = A \multimap A \multimap B, S = B \multimap !A$. A possible typing for $\mathtt{M}$ is $\mathtt{y}:C, \mathtt{s}:S, \mathtt{w}:B \vdash_{\mathrm{L}} \mathtt{y}((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}):B$ as proved by the following

TABLE 2. SLL_λ

$$\frac{}{\mathtt{x}:A\vdash_{\mathrm{L}}\mathtt{x}:A}\ (Ax) \qquad \frac{\Gamma\vdash_{\mathrm{L}}\mathtt{M}:A \quad \Delta,\mathtt{x}:A\vdash_{\mathrm{L}}\mathtt{N}:B \quad \Gamma\#\Delta}{\Gamma,\Delta\vdash_{\mathrm{L}}\mathtt{N}[\mathtt{M}/\mathtt{x}]:B}\ (cut)$$

$$\frac{\Gamma,\mathtt{x}:A\vdash_{\mathrm{L}}\mathtt{M}:B}{\Gamma\vdash_{\mathrm{L}}\lambda\mathtt{x}.\mathtt{M}:A\multimap B}\ (\multimap R) \qquad \frac{\Gamma,\mathtt{x}_0:A,...,\mathtt{x}_n:A\vdash_{\mathrm{L}}\mathtt{M}:B}{\Gamma,\mathtt{x}:!A\vdash_{\mathrm{L}}\mathtt{M}[\mathtt{x}/\mathtt{x}_0,...,\mathtt{x}/\mathtt{x}_n]:B}\ (m)$$

$$\frac{\Gamma\vdash_{\mathrm{L}}\mathtt{M}:A \quad \mathtt{x}:B,\Delta\vdash_{\mathrm{L}}\mathtt{N}:C \quad \Gamma\#\Delta \quad \mathtt{y}\text{ fresh}}{\Gamma,\mathtt{y}:A\multimap B,\Delta\vdash_{\mathrm{L}}\mathtt{N}[\mathtt{y}\mathtt{M}/\mathtt{x}]:C}\ (\multimap L) \qquad \frac{\Gamma\vdash_{\mathrm{L}}\mathtt{M}:A}{!\Gamma\vdash_{\mathrm{L}}\mathtt{M}:!A}\ (sp)$$

$$\frac{\Gamma\vdash_{\mathrm{L}}\mathtt{M}:A \quad (*)}{\Gamma\vdash_{\mathrm{L}}\mathtt{M}:\forall\alpha.A}\ (\forall R) \qquad \frac{\Gamma,\mathtt{x}:A[B/\alpha]\vdash_{\mathrm{L}}\mathtt{M}:C}{\Gamma,\mathtt{x}:\forall\alpha.A\vdash_{\mathrm{L}}\mathtt{M}:C}\ (\forall L)$$

$\Gamma\#\Delta$ iff $\mathrm{dom}(\Gamma)\cap\mathrm{dom}(\Delta)\equiv\emptyset$ $\qquad$ $(*)$ α not free in Γ.

(incomplete) derivation Π:

$$\dfrac{\dfrac{\mathtt{s}:S\vdash_{\mathrm{L}}\lambda\mathtt{z}.\mathtt{s}\mathtt{z}:S \quad \mathtt{t}:S,\mathtt{w}:B\vdash_{\mathrm{L}}\mathtt{t}\mathtt{w}:!A}{\Sigma\rhd\mathtt{s}:S,\mathtt{w}:B\vdash_{\mathrm{L}}(\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}:!A}\ (cut) \quad \dfrac{\mathtt{y}:C,\mathtt{r}:A,\mathtt{l}:A\vdash_{\mathrm{L}}\mathtt{y}\mathtt{r}\mathtt{l}:B}{\mathtt{y}:C,\mathtt{x}:!A\vdash_{\mathrm{L}}\mathtt{y}\mathtt{x}\mathtt{x}:B}\ (m)}{\mathtt{y}:C,\mathtt{s}:S,\mathtt{w}:B\vdash_{\mathrm{L}}\mathtt{y}\mathtt{x}\mathtt{x}[(\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}/\mathtt{x}]\equiv\mathtt{y}((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}):B}\ (cut)$$

Clearly we have:

$$\mathtt{y}((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})\to_\beta\mathtt{y}(\mathtt{s}\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w})$$

hence we want to type the reduced term in the same context:

$$\mathtt{y}:C,\mathtt{s}:S,\mathtt{w}:B\vdash_{\mathrm{L}}\mathtt{y}((\mathtt{s}\mathtt{w})((\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}):B$$

but in fact this is not a derivable judgement in SLL_λ.

The problem is that $\mathtt{M}$ contains two identical redexes $(\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}$, while in Π there is just one subderivation Σ with subject $(\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}$. Σ has a modal conclusion, but a non modal context, so it cannot be duplicated. Hence β-reducing only one occurrence of $(\lambda\mathtt{z}.\mathtt{s}\mathtt{z})\mathtt{w}$ in $\mathtt{M}$ would not correspond to a correct logical proof since every cut-elimination step would instead reduce both the redexes at the same time. This means that in SLL_λ a cut-elimination step does not correspond to a β-reduction and the polynomial bound on the logic is not inherited by the language.

Let us analyze this phenomenon in more depth. In order to do this note that we can distinguish three classes of SLL_λ derivations. Consider a derivation Π in SLL_λ with conclusion $\Gamma\vdash_{\mathrm{L}}\mathtt{M}:A$. The formula A can be either modal or not. In the latter case we say that Π is a *linear* derivation. Otherwise we can distinguish two cases. If Π can be transformed by commutations of rules in a derivation Π' ending by a rule (sp) then we say that Π is *duplicable*. Otherwise we say that it is *shareable*. Note that in other words Π is duplicable if it corresponds to a !-box in the corresponding proof-net of SLL [17].

In a decorated sequent calculus system, like SLL_λ, β-reduction is usually considered the counterpart, in terms, of the cut rule of the logic. So consider an occurrence of a rule (cut) as:

$$\frac{\Pi \rhd \Gamma \vdash_{\mathrm{L}} \mathtt{M}: A \quad \Sigma \rhd \Delta, \mathtt{x}: A \vdash_{\mathrm{L}} \mathtt{N}: B}{\Gamma, \Delta \vdash_{\mathrm{L}} \mathtt{N}[\mathtt{M}/\mathtt{x}]: B}\ (cut)$$

We can split it in three distinct rules, according to the class of Π.

Linear cut

$$\frac{\Pi \rhd \Gamma \vdash_{\mathrm{L}} \mathtt{M}: A \quad \Sigma \rhd \Delta, \mathtt{x}: A \vdash_{\mathrm{L}} \mathtt{N}: B \quad \Pi \text{ linear}}{\Gamma, \Delta \vdash_{\mathrm{L}} \mathtt{N}[\mathtt{M}/\mathtt{x}]: \mathtt{B}}\ (L\ cut)$$

It corresponds to a *linear* substitution. In case there is a subterm of N of the shape xQ, for some Q (denoted by $\mathtt{N} \equiv \mathtt{N}[\mathtt{xQ}]$) and $\mathtt{M} \equiv \lambda\mathtt{x}.\mathtt{P}$, it generates exactly one β-redex and the cut-elimination corresponds to one β-reduction.

Duplication cut

$$\frac{\Pi \rhd !\Gamma' \vdash_{\mathrm{L}} \mathtt{M}: !C \quad \Delta, \mathtt{x}: !C \vdash_{\mathrm{L}} \mathtt{N}: B \qquad A \equiv !C \quad \Gamma = !\Gamma' \quad \Pi \text{ duplicable}}{!\Gamma', \Delta \vdash_{\mathrm{L}} \mathtt{N}[\mathtt{M}/\mathtt{x}]: B}\ (D\ cut)$$

It corresponds to a substitution where the proof Π is *copied* n times, where n is the number of occurrences of $\mathtt{x}$ in $\mathtt{N}$. In case $\mathtt{N} \equiv \mathtt{N}[\mathtt{xQ}]$ and $\mathtt{M} \equiv \lambda\mathtt{x}.\mathtt{P}$, it generates n β-redexes. The elimination of such a kind of cut generates n further cuts (see Definition 2.4) and the elimination of each one of these cuts corresponds to the reduction of a β-redex.

Sharing cut

$$\frac{\Pi \rhd \Gamma \vdash_{\mathrm{L}} \mathtt{M}: !C \quad \Delta, \mathtt{x}: !C \vdash_{\mathrm{L}} \mathtt{N}: B \quad A \equiv !C \quad \Pi \text{ shareable}}{\Gamma, \Delta \vdash_{\mathrm{L}} \mathtt{N}[\mathtt{M}/\mathtt{x}]: C}\ (S\ cut)$$

It corresponds to a substitution where the proof Π is not duplicated, but *shared* n times, where n is the number of occurrences of $\mathtt{x}$ in $\mathtt{N}$. In case $\mathtt{N} \equiv \mathtt{N}[\mathtt{xQ}]$ and $\mathtt{M} \equiv \lambda\mathtt{x}.\mathtt{P}$, it generates n β-redexes but a single cut-elimination step corresponds to β-reducing in parallel all of them.
So $(S\ cut)$ can break the correspondence between cut elimination and β-reduction.

The problem described above is not new, and all the type assignment systems for λ-calculus derived from Linear Logic need to deal with it. Until now the proposed solutions follow three different paths, all based on a natural deduction definition of the type assignment.
The first one, proposed in [19], based on Intuitionistic Linear Logic, explicitly checks the duplicability condition before performing a normalization step. So in the resulting language (which is a fully typed λ-calculus) the set of redexes is a proper subset of the set of classical β-redexes.

TABLE 3. Essential Soft Linear Logic.

$$\frac{}{A\vdash_{\mathrm{E}} A}\ (Ax) \qquad \frac{\Gamma\vdash_{\mathrm{E}}\tau \quad \Delta,\tau\vdash_{\mathrm{E}}\sigma}{\Gamma,\Delta\vdash_{\mathrm{E}}\sigma}\ (cut) \qquad \frac{\Gamma,A[B/\alpha]\vdash_{\mathrm{E}}\sigma}{\Gamma,\forall\alpha.A\vdash_{\mathrm{E}}\sigma}\ (\forall L)$$

$$\frac{\Gamma\vdash_{\mathrm{E}}\tau \quad A,\Delta\vdash_{\mathrm{E}}\sigma}{\Gamma,\tau\multimap A,\Delta\vdash_{\mathrm{E}}\sigma}\ (\multimap L) \qquad \frac{\Gamma,\sigma\vdash_{\mathrm{E}} A}{\Gamma\vdash_{\mathrm{E}}\sigma\multimap A}\ (\multimap R) \qquad \frac{\Gamma\vdash_{\mathrm{E}} A}{\Gamma\vdash_{\mathrm{E}}\forall\alpha.A}\ (\forall R)$$

$$\frac{\Gamma\vdash_{\mathrm{E}}\sigma}{!\Gamma\vdash_{\mathrm{E}}!\sigma}\ (sp) \qquad \frac{\Gamma,\tau^{(n)}\vdash_{\mathrm{E}}\sigma \quad n\geq 0}{\Gamma,!\tau\vdash_{\mathrm{E}}\sigma}\ (m)$$

In [4] a type assignment for λ-calculus, based on Light Affine Logic, is designed where the modality ! is no more explicit. In fact there are two arrows, a linear and an intuitionistic one, whose eliminations reflect the linear and the duplication cuts, respectively.

In [7], a type assignment for the λ-calculus, based on Elementary Affine Logic, is designed. There the authors use the call-by-value λ-calculus, where the restricted definition of reduction implies that subterms of the shape $(\lambda \mathtt{x}.\mathtt{M})\mathtt{N}$, when generated by an $(S\ cut)$, are not call-by-value β-redexes.

All the approaches need a careful control of the context, which technically has been realized by splitting it in different parts, collecting respectively the linear and modal assumptions (in case of [7] a further context is needed).

Here we want to explore a different approach starting from the following considerations. Let Π be a SLL_λ derivation with subject $\mathtt{M}$. If either Π or some proof in which Π rewrites during the cut-elimination procedure contains some $(S\ cut)$ rule, the number of β-reductions in the normalization of $\mathtt{M}$ can be greater than the number of cut-elimination steps in the normalization of Π. So the typing does not induce any property on the complexity of $\mathtt{M}$. Conversely, if neither Π nor any proof in which Π rewrites during the cut-elimination procedure does contain $(S\ cut)$, then the number of cut-elimination steps in the normalization of Π is greater or equal to the number of β-reductions in the normalization of $\mathtt{M}$ and then the polynomial bound for $\mathtt{M}$ follows.
So we restrict SLL in such a way that S-cuts are forbidden and the polynomial properties are preserved. Just erasing the rule $(S\ cut)$ is not enough, since the cut elimination procedure could create new $(S\ cut)$ rules. So we need to restrict both the rules and the formulae.

4 Essential Soft Linear Logic

In this section we will present a light logic, ESLL, whose set of proofs is a proper subset of proofs of SLL, and a natural deduction version of it. Both preserve the polytime correctness and completeness. The key point is that every ESLL proof does not contain $(S\ cut)$, and moreover $(S\ cuts)$ cannot be created during the normalization precedure. In order to obtain this property, we need to restrict the syntax of types not allowing modal types both in the right of a $\multimap$ and under a $\forall$, and restricting axioms to introduce only non modal types.

Definition 4.1

(i) The set **EF** of *essential soft formulae* is defined as follows:

$$A ::= \alpha \mid \sigma \multimap A \mid \forall\alpha.A \quad \text{(Linear Formulae)}$$
$$\sigma ::= A \mid !\sigma$$

where α (β) ranges over a countable set of variables. Linear formulae are ranged over by A, B, C, and formulae by σ, τ, ζ. The symbol $\equiv$ denotes the syntactical equality (modulo renaming of bound variables).

(ii) A context is a multiset of formulae. By an abuse of notation, contexts are ranged over by Γ, Δ. The notation $!\Gamma$ is a short for $\{!\sigma \mid \sigma \in \Gamma\}$.

(iii) ESLL proves sequents of the shape $\Gamma \vdash_{\mathrm{E}} \sigma$ where Γ is a context and σ is an essential soft formula. The rules are given in Table 3, where, as usual, the rule $(\forall R)$ has the side condition that α must not be free in Γ. The notation $\vdash_{\mathrm{E}} \sigma$ is a short for $\emptyset \vdash_{\mathrm{E}} \sigma$.

(iv) Derivations are denoted by $\Pi, \Sigma, \Phi, \Psi, \Theta$. $\Pi \rhd \Gamma \vdash_{\mathrm{E}} \sigma$ denotes a derivation Π with conclusion $\Gamma \vdash_{\mathrm{E}} \sigma$.

Essential Soft Linear Logic is a subsystem of Soft Linear Logic.

Lemma 4.2

The set of proofs of ESLL is a proper subset of proofs of SLL.

Proof. Trivial. ∎

As corollary, we have the following theorem.

Theorem 4.3

ESLL enjoys cut-elimination. In particular every ESLL proof normalizes in at most $2 \times |\Pi|^{3(\mathrm{d}(\Pi)+1)}$ cut elimination steps.

Proof. The cut-elimination procedure for SLL can be applied, with the same bound, since Lemma 4.2. ∎

The absence of (S *cut*) in ESLL proof is assured by the following property.

Property 4.4

$\Pi \rhd \Gamma \vdash_{\mathrm{E}} !\sigma$ implies all premises in Γ are modal.

Proof. By induction on Π. In the case Π ends by (sp) the conclusion follows directly. In the case Π ends either by (m) or (*cut*) rule, the conclusion follows by induction hypothesis. The other cases are not allowed since the syntactical conditions on the formation rules. ∎

4.1 ESLL in (quasi) Natural Deduction

A type assignment system for λ-calculus is in general defined in natural deduction style, since in this way terms are built inductively by the rules of the system, and proofs can be easily carried out by induction on the size of terms. In order to design a type assignment system with such property, we tranform ESLL in a style similar to natural deduction, in the sense that all rules are in natural deduction, but rules (m) and (sp), which are non local. But, as we will show in the sequel, in the term decoration of the new system the application of each one of these two rules does not change the size of the subject, so it will be possible to reason about the type assignment system by induction on the size of terms,

TABLE 4. ESLL in (quasi) natural deduction style

$$\frac{}{A\vdash_{\mathrm{N}} A}\ (Ax) \qquad \frac{\Gamma\vdash_{\mathrm{N}} A \quad \alpha\notin \mathrm{FTV}(\Gamma)}{\Gamma\vdash_{\mathrm{N}} \forall\alpha.A}\ (\forall I)$$

$$\frac{\Gamma,\sigma\vdash_{\mathrm{N}} A}{\Gamma\vdash_{\mathrm{N}} \sigma\multimap A}\ (\multimap I) \qquad \frac{\Gamma\vdash_{\mathrm{N}} \sigma\multimap A \quad \Delta\vdash_{\mathrm{N}}\sigma \quad \Gamma\#\Delta}{\Gamma,\Delta\vdash_{\mathrm{N}} A}\ (\multimap E)$$

$$\frac{\Gamma,\tau^{(n)}\vdash_{\mathrm{N}}\sigma \quad n\geq 0}{\Gamma,!\tau\vdash_{\mathrm{N}}\sigma}\ (m) \qquad \frac{\Gamma\vdash_{\mathrm{N}}\sigma}{!\Gamma\vdash_{\mathrm{N}}!\sigma}\ (sp) \qquad \frac{\Gamma\vdash_{\mathrm{N}}\forall\alpha.B}{\Gamma\vdash_{\mathrm{N}} B[A/\alpha]}\ (\forall E)$$

and this is sufficient for our aims. In fact, a true natural deduction version of ESLL would be quite involved, since the modality !. The (quasi) natural deduction version of ESLL is a system named NESLL, and we will prove that it preserves the good properties of ESLL, with respect to the complexity of the normalization procedure. The system NESLL proves judgements of the shape $\Gamma\vdash_{\mathrm{N}}\sigma$, where Γ is a context and σ is an essential soft formula, according to Definition 4.1. The rules of the system are given in Table 4. We extend to NESLL all notations of Definition 4.1.

We now prove some important properties of NESLL, which will be useful in the sequel. First, let us observe that an analogous of Property 4.4 holds:

PROPERTY 4.5
(i) $\Pi\rhd\Gamma\vdash_{\mathrm{N}}!\sigma$ implies that all premises in Γ are modal.
(ii) $\Pi\rhd\Gamma\vdash_{\mathrm{N}}!\sigma$ implies that Π is composed by a proof $\Pi':!\Delta\vdash_{\mathrm{N}}!\sigma$ ending with an (sp) rule, followed by some applications of (m) rule.

PROOF. Both points can be easily proved by induction on Π. ∎

NESLL enjoys the substitution property.

LEMMA 4.6 (Substitution)
Let $\Pi\rhd\Gamma,\tau\vdash_{\mathrm{N}}\sigma$ and $\Sigma\rhd\Delta\vdash_{\mathrm{N}}\tau$. Then there is $S(\Sigma,\Pi)\rhd\Gamma,\Delta\vdash_{\mathrm{N}}\sigma$.

PROOF. The proof is given by induction on Π. In case Π is an axiom, $S(\Sigma,\Pi)=\Sigma$. The cases where Π ends by an application of the rules $(\multimap I),(\multimap E),(\forall I),(\forall E)$ $S(\Sigma,\Pi)$ is defined by induction. Let the last rule of Π be (m). In case the active formula is different from τ, then the proof follows by induction. Otherwise, let Π end by:

$$\frac{\Pi_1:\Gamma,\tau_1^{(n)}\vdash_{\mathrm{N}}\sigma}{\Gamma,\tau\vdash_{\mathrm{N}}\sigma}\ (m)$$

If τ has been introduced by a multiplexor of rank 0 then $S(\Sigma,\Pi)$ is $S(\Sigma,\Pi_1)$ followed by some applications of (m) rule in order to recover the context Δ which is modal by Property 4.5.(i). Otherwise, by Property 4.5.(ii) Σ is composed by a subderivation ending with the rule

$$\frac{\Sigma'\rhd\Delta_1\vdash_{\mathrm{N}}\tau_1}{!\Delta_1\vdash_{\mathrm{N}}\tau}\ (sp)$$

followed by a sequence δ of (m) rules recovering $\Delta \vdash_{\mathrm{N}} \tau$. By induction we can build $S(\Sigma', \Pi)$, $S(\Sigma', S(\Sigma', \Pi))$,..., $S(\Sigma', \underbrace{S(\Sigma', (S(\Sigma', ..., S(\Sigma'}_{n}, \Pi))))))$, the last proving $\Gamma, \underbrace{\Delta_1, ..., \Delta_1}_{n} \vdash_{\mathrm{N}} \sigma$. Then (Σ, Π) can be obtained from $\Gamma, \underbrace{\Delta_1, ..., \Delta_1}_{n} \vdash_{\mathrm{N}} \sigma$ by applying a sequence of (m) rules, in order to obtain $\Gamma, !\Delta_1 \vdash_{\mathrm{N}} \sigma$, followed by δ. The case Π ends by rule (sp) is similar. ∎

The normalization steps for NESLL are defined in the following.

Definition 4.7

- $(\multimap)$-normalization step.
  A $(\multimap)$-redex in Π is a subproof of the shape:

$$\dfrac{\dfrac{\Psi \rhd \Gamma, \sigma \vdash_{\mathrm{N}} A}{\Gamma \vdash_{\mathrm{N}} \sigma \multimap A}\ (\multimap I) \qquad \Phi \rhd \Delta \vdash_{\mathrm{N}} \sigma}{\Gamma, \Delta \vdash_{\mathrm{N}} A}\ (\multimap E)$$

  and its reduct is $S(\Phi, \Psi)$, defined in the Substitution Lemma.
- $(\forall)$-normalization step.
  A $(\forall)$-redex in Π is a subproof of the shape:

$$\dfrac{\dfrac{\Psi \rhd \Gamma \vdash_{\mathrm{N}} A}{\Gamma \vdash_{\mathrm{N}} \forall \alpha. A}\ (\forall I)}{\Gamma \vdash_{\mathrm{N}} A[B/\alpha]}\ (\forall E)$$

  and its reduct is the proof Ψ, where all free occurrences of α has been replaced by B.

In order to prove that NESLL and ESLL have the same derivability power, we start by proving the implication from ESLL to NESLL.

Theorem 4.8
$\Gamma \vdash_{\mathrm{E}} A$ implies $\Gamma \vdash_{\mathrm{N}} A$

Proof. By induction on the derivation Π proving $\Gamma \vdash_{\mathrm{E}} A$. The base case is trivial. The cases where Π end by $(\multimap R), (m), (sp)$ and $(\forall R)$ follow directly by induction hypothesis. The case Π ends by the (cut) rule follows directly by induction hypothesis and Lemma 4.6. Consider the case Π ends as:

$$\dfrac{\Gamma \vdash_{\mathrm{E}} \tau \quad A, \Delta \vdash_{\mathrm{E}} \sigma}{\Gamma, \tau \multimap A, \Delta \vdash_{\mathrm{E}} \sigma}\ (\multimap L)$$

By induction hypothesis we have a derivation with conclusion $\Gamma \vdash_{\mathrm{N}} \tau$ hence we can construct a derivation ending as:

$$\dfrac{\dfrac{}{\tau \multimap A \vdash_{\mathrm{N}} \tau \multimap A}\ (Ax) \qquad \Gamma \vdash_{\mathrm{N}} \tau}{\Gamma, \tau \multimap A \vdash_{\mathrm{N}} A}\ (\multimap E)$$

and since by induction hypothesis we also have $A, \Delta \vdash_{\mathrm{N}} \sigma$, by applying Lemma 4.6 the conclusion follows.

Consider the case Π ends by the rule:

$$\frac{\Gamma, A[B/\alpha] \vdash_{\mathrm{E}} \sigma}{\Gamma, \forall\alpha.A \vdash_{\mathrm{E}} \sigma}\ (\forall L)$$

By induction hypothesis we have a derivation with conclusion $\Gamma, A[B/\alpha] \vdash_{\mathrm{N}} \sigma$ and since we also have a derivation ending as:

$$\frac{\dfrac{}{\forall\alpha.A \vdash_{\mathrm{N}} \forall\alpha.A}\ (Ax)}{\forall\alpha.A \vdash_{\mathrm{N}} A[B/\alpha]}\ (\forall E)$$

by applying Lemma 4.6 the conclusion follows. ∎

We now need to prove that NESLL implies ESLL. Since there is a one to many correspondence between NESLL proofs and ESLL proofs, we will show a translation preserving the good properties of ESLL with respect to the normalization time. So we prove at the same time the equivalence between the two systems and the polynomial soundness of NESLL. In order to do this, we will use the notions of *size*, *rank* and *degree* of a proof, defined in Definition 2.2, which can be extended to NESLL in the obvious way.

Lemma 4.9
There is a translation T, from NESLL proofs to ESLL proofs, such that, if $\Pi \rhd \Gamma \vdash_{\mathrm{N}} \sigma$, then $T(\Pi) \rhd \Gamma \vdash_{\mathrm{E}} \sigma$, and moreover T preserves the measures, i.e., $\mathtt{rk}(\Pi) = \mathtt{rk}(T(\Pi))$, $\mathtt{d}(\Pi) = \mathtt{d}(T(\Pi))$ and $|T(\Pi)| \leq 3 \times (|\Pi|)$.

Proof. T is defined by induction on Π. In the (Ax) case, T is the identity. Otherwise, let Π be

$$\frac{\Pi' \rhd \Gamma \vdash_{\mathrm{N}} \sigma}{\Gamma' \vdash_{\mathrm{N}} \sigma'}\ (R)$$

If R is (m) or (sp), then $T(\Pi)$ is $T(\Pi')$ followed by (R). If R is $(\multimap I)$ or $(\forall I)$, then $T(\Pi)$ is $T(\Pi')$ followed by $(\multimap R)$ or $(\forall R)$. If Π ends by the rule:

$$\frac{\Sigma_1 \rhd \Delta \vdash_{\mathrm{N}} \tau \multimap A \quad \Sigma_2 \rhd \Gamma \vdash_{\mathrm{N}} \tau}{\Gamma, \Delta \vdash_{\mathrm{N}} A}\ (\multimap E)$$

then $T(\Pi)$ is

$$\frac{T(\Sigma_1) \rhd \Delta \vdash_{\mathrm{E}} \tau \multimap A \qquad \dfrac{T(\Sigma_2) \rhd \Gamma \vdash_{\mathrm{E}} \tau \quad \dfrac{}{A \vdash_{\mathrm{E}} A}\ (Ax)}{\Gamma, \tau \multimap A \vdash_{\mathrm{E}} A}\ (\multimap L)}{\Gamma, \Delta \vdash_{\mathrm{E}} A}\ (cut)$$

If Π ends by the rule:

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{N}} \forall\alpha.A}{\Gamma \vdash_{\mathrm{N}} A[B/\alpha]}\ (\forall E)$$

then $T(\Pi)$ is

$$\dfrac{T(\Sigma)\rhd\Gamma\vdash_{\mathrm{E}}\forall\alpha.A \qquad \dfrac{\dfrac{}{A[B/\alpha]\vdash_{\mathrm{E}}A[B/\alpha]}\,(Ax)}{\forall\alpha.A\vdash_{\mathrm{E}}A[B/\alpha]}\,(\forall L)}{\Gamma\vdash_{\mathrm{E}}A[B/\alpha]}\,(cut)$$

Now for each Π it is easy to verify that $\mathtt{rk}(\Pi)=\mathtt{rk}(T(\Pi))$, $\mathtt{d}(\Pi)=\mathtt{d}(T(\Pi))$. Moreover the size of $T(\Pi)$ is equal to the sum of the size of $|\Pi|$ and two times the number of $(\multimap E)$ and $(\forall E)$ in Π. So in particular $|T(\Pi)|\leq 3\times|(\Pi)|$. ∎

The following lemma shows the key property, that allows to extend to NESLL the complexity results obtained for ESLL.

Lemma 4.10
Let Π be a NESLL proof, which reduces to Π' in n normalization steps. Then $T(\Pi)$ reduces to $T(\Pi')$ in $m\geq n$ cut-elimination steps.

Proof. It is sufficient to prove the lemma for $n=1$. If the normalization step reduces a $(\forall)$-redex, then the proof can be carried out easily by induction on the proof which is the premise of the $(\forall I)$ rule. Consider the case of a $(\multimap)$-redex, so let Π contain a subproof of the shape:

$$\dfrac{\dfrac{\Psi\rhd\Gamma,\sigma\vdash_{\mathrm{N}}A}{\Gamma\vdash_{\mathrm{N}}\sigma\multimap A}\,(\multimap I) \qquad \Phi\rhd\Delta\vdash_{\mathrm{N}}\sigma}{\Gamma,\Delta\vdash_{\mathrm{N}}A}\,(\multimap E)$$

and Π' is obtained by replacing this subproof by $S(\Phi,\Psi)$. Then $T(\Pi)$ contain a subproof:

$$\dfrac{\dfrac{T(\Psi)\rhd\Gamma,\sigma\vdash_{\mathrm{E}}A}{\Gamma\vdash_{\mathrm{E}}\sigma\multimap A}\,(\multimap R) \qquad \dfrac{T(\Phi)\rhd\Delta\vdash_{\mathrm{E}}\sigma \qquad \dfrac{}{A\vdash_{\mathrm{E}}A}\,(Ax)}{\Delta,\tau\multimap A\vdash_{\mathrm{E}}\sigma}\,(\multimap L)}{\Gamma,\Delta\vdash_{\mathrm{E}}A}\,(cut)$$

Let R and R' be respectively the last applied rule of $T(\Psi)$ and $T(\Phi)$. Note that:

1. neither R nor R' can be a left introduction rule (by definition of T), and in particular R cannot be (sp) (by definition of essential soft formulae).

By two cut elimination steps we obtain:

$$\dfrac{T(\Phi)\rhd\Delta\vdash_{\mathrm{E}}\sigma \qquad T(\Psi)\rhd\Gamma,\sigma\vdash_{\mathrm{E}}A}{\Gamma,\Delta\vdash_{\mathrm{E}}A}\,(cut)$$

and we need to prove that this reduces to $S(\Phi,\Psi)$, so $T(\Pi)$ reduces to $T(\Pi')$. The proof is carried out by induction on Ψ. All possible cases follow by induction, but the case of $(sp)-(m)$ cut. In this case the redex is:

$$\dfrac{\dfrac{\Theta_1\rhd\Delta'\vdash_{\mathrm{E}}\tau}{T(\Phi)\rhd!\Delta'\vdash_{\mathrm{E}}!\tau}\,(sp) \qquad \dfrac{\Theta_2\rhd\Gamma,\tau^{(n)}\vdash_{\mathrm{E}}A}{T(\Psi)\rhd\Gamma,!\tau\vdash_{\mathrm{E}}A}\,(m)}{!\Delta',\Gamma\vdash_{\mathrm{E}}A}\,(cut)$$

where $\sigma \equiv !\tau$ and $\Delta \equiv !\Delta'$. Note that, by definition of T, there are two NESLL proofs Ψ' and Φ' such that $\Theta_1 = T(\Phi')$ and $\Theta_2 = T(\Psi')$.
Eliminating this cut we obtain:

$$\dfrac{\dfrac{T(\Phi')\rhd\Delta'\vdash_{\mathrm{E}}\tau \qquad \dfrac{T(\Psi')\rhd\Gamma,\tau^{(n)}\vdash_{\mathrm{E}}A}{\Delta'^{(n-1)},\tau,\Gamma\vdash_{\mathrm{E}}A}\,(cut)}{\dfrac{\Delta'^{(n)},\Gamma\vdash_{\mathrm{E}}A}{!\Delta',\Gamma\vdash_{\mathrm{E}}A}\,(m)}\,(cut)$$

By induction the elimination of all these cuts generates $\underbrace{S(T(\Phi'),S(T(\Phi'),...,S(T(\Phi')}_{n},T(\Psi'))$.
This, followed by the sequence of applications of (m) rules, is, by definition, $S(T(\Phi),T(\Psi))$.
So, replacing it to the redex in Π, we obtain $T(\Pi')$. ∎

We can now prove the following.

Theorem 4.11
Let Π be a NESLL proof. Then Π is strongly normalizing and the number of normalization steps to normal form is $O(|\Pi|^{3\times(\mathrm{d}(\Pi)+1)})$.

Proof. By Lemma 4.10 and Theorem 4.3. ∎

5 The Soft Type Assignment System

Finally a type assignment system for λ-calculus can be obtained, just decorating, in a standard way, NESLL proofs by λ-terms. We will prove that such system inherits from NESLL all the desired properties.

Definition 5.1

i) Terms of λ-calculus are defined by the following grammar:

$$\mathtt{M},\mathtt{N},\mathtt{Q} ::= \mathtt{x} \mid \mathtt{M}\mathtt{M} \mid \lambda\mathtt{x}.\mathtt{M}$$

where $\mathtt{x}$ ranges over a countable set of variables.

ii) The reduction relation $\rightarrow_\beta$ is the contextual closure of the following rule:

$$(\lambda\mathtt{x}.\mathtt{M})\mathtt{N} \rightarrow_\beta \mathtt{M}[\mathtt{N}/\mathtt{x}]$$

where, as usual, $\mathtt{M}[\mathtt{N}/\mathtt{x}]$ is the capture free substitution of $\mathtt{N}$ to all the free occurrences of $\mathtt{x}$ in $\mathtt{M}$. $\rightarrow_\beta^*$ is the reflexive and transitive closure of $\rightarrow_\beta$.

iii) A context is a set of assumptions of the shape $\mathtt{x}:\sigma$, where $\mathtt{x}$ is a variable and σ is an essential soft type. Variables in a context are all distinct. By abuse of notation, contexts are ranged over by Γ,Δ,Θ.

iv) The Soft Type Assignment System (STA) proves statements of the shape:

$$\Gamma\vdash\mathtt{M}:\sigma$$

where Γ is a context, $\mathtt{M}$ is a term of λ-calculus, and σ is an essential soft type. The rules are given in Table 5.

TABLE 5. The Soft Type Assignment system STA

$$\frac{}{\mathtt{x}:A\vdash\mathtt{x}:A}\ (Ax)$$

$$\frac{\Gamma,\mathtt{x}:\sigma\vdash\mathtt{M}:A}{\Gamma\vdash\lambda\mathtt{x}.\mathtt{M}:\sigma\multimap A}\ (\multimap I)\qquad \frac{\Gamma\vdash\mathtt{M}:\sigma\multimap A\quad\Delta\vdash\mathtt{N}:\sigma\quad\Gamma\#\Delta}{\Gamma,\Delta\vdash\mathtt{MN}:A}\ (\multimap E)$$

$$\frac{\Gamma,\mathtt{x}_1:\sigma,\ldots,\mathtt{x}_n:\sigma\vdash\mathtt{M}:\mu}{\Gamma,\mathtt{x}:!\sigma\vdash\mathtt{M}[\mathtt{x}/\mathtt{x}_1,\cdots,\mathtt{x}/\mathtt{x}_n]:\mu}\ (m)\qquad \frac{\Gamma\vdash\mathtt{M}:\sigma}{!\Gamma\vdash\mathtt{M}:!\sigma}\ (sp)$$

$$\frac{\Gamma\vdash\mathtt{M}:A\quad\alpha\notin\mathrm{FTV}(\Gamma)}{\Gamma\vdash\mathtt{M}:\forall\alpha.A}\ (\forall I)\qquad \frac{\Gamma\vdash\mathtt{M}:\forall\alpha.B}{\Gamma\vdash\mathtt{M}:B[A/\alpha]}\ (\forall E)$$

5.1 Polynomial Time Soundness

STA is sound for polynomial time. In particular in STA a $\multimap$-normalization step corresponds to a β-reduction, so the polynomial soundness of NESLL implies directly the polynomial soundness of STA. Nevertheless the bound is expressed with respect to the size of the typing proof, while we are interested in giving it with respect to the size of the term. In order to do this we will introduce two new measures, the λ-rank $\overline{\mathtt{rk}}$ and the λ-weight $\overline{\mathtt{w}}$ counting the number of effectively contracted variables and the number of $\multimap$-reduction steps respectively.

DEFINITION 5.2

- The size $|\mathtt{M}|$ of a term $\mathtt{M}$ is defined as $|\mathtt{x}|=1,\ |\lambda\mathtt{x}.\mathtt{M}|=|\mathtt{M}|+1,\ |\mathtt{MN}|=|\mathtt{M}|+|\mathtt{N}|+1$.
- The λ-*rank* of a rule (m):

$$\frac{\Gamma,\mathtt{x}_1:\tau,\ldots,\mathtt{x}_n:\tau\vdash\mathtt{M}:\sigma}{\Gamma,\mathtt{x}:!\tau\vdash\mathtt{M}[\mathtt{x}/\mathtt{x}_1,\ldots,\mathtt{x}/\mathtt{x}_n]:\sigma}\ (m)$$

  is the number $k\leq n$ of variables $\mathtt{x}_i$ such that $\mathtt{x}_i\in\mathrm{FV}(\mathtt{M})$. Let r be the the maximum λ-rank of a rule (m) in Π. The λ-rank $\overline{\mathtt{rk}}(\Pi)$ of Π is the maximum between 1 and r.
- The λ-weight $\overline{\mathtt{w}}(\Pi,r)$ of Π with respect to r is defined inductively as follows.
  - If the last applied rule is (Ax) then $\overline{\mathtt{w}}(\Pi,r)=1$.
  - If the last applied rule is $(\multimap I)$ with premise a derivation Σ, then $\overline{\mathtt{w}}(\Pi,r)=\overline{\mathtt{w}}(\Sigma,r)+1$.
  - If the last applied rule is (sp) with premise a derivation Σ, then $\overline{\mathtt{w}}(\Pi,r)=r\overline{\mathtt{w}}(\Sigma,r)$.
  - If the last applied rule is $(\multimap E)$ with premises Σ and Φ then $\overline{\mathtt{w}}(\Pi,r)=\overline{\mathtt{w}}(\Sigma,r)+\overline{\mathtt{w}}(\Phi,r)+1$.
  - In every other case $\overline{\mathtt{w}}(\Pi,r)=\overline{\mathtt{w}}(\Sigma,r)$ where Σ is the unique premise derivation.

The previously introduced measures are related each other as shown explicitly by the following lemma:

LEMMA 5.3
Let $\Pi\rhd\Gamma\vdash\mathtt{M}:\sigma$. Then:

1. $\overline{\mathtt{rk}}(\Pi)\leq|\mathtt{M}|\leq|\Pi|$.
2. $\overline{\mathtt{w}}(\Pi,r)\leq r^{\mathtt{d}(\Pi)}\overline{\mathtt{w}}(\Pi,1)$
3. $\overline{\mathtt{w}}(\Pi,1)=|\mathtt{M}|$.

Proof. Easy. ∎

The following is the weighted version of Lemma 4.6

Lemma 5.4 (Weighted Substitution)
Let $\Pi \rhd \Gamma, \mathtt{x}:\tau \vdash \mathtt{M}:\sigma$ and $\Sigma \rhd \Delta \vdash \mathtt{N}:\tau$. Then $\Phi \rhd \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}]:\sigma$ and $\overline{\mathtt{w}}(\Phi, r) \leq \overline{\mathtt{w}}(\Pi) + \overline{\mathtt{w}}(\Sigma)$.

Proof. It follows easily by an inspection of how the λ-weight changes in the proof of Lemma 4.6. ∎

From the weighted substitution it follows that the weight decreases when a $\multimap$-normalization step is performed.

Lemma 5.5
Let $\Pi \rhd \Gamma \vdash \mathtt{M}:\sigma$ and Π rewrites to $\Pi' \rhd \Gamma \vdash \mathtt{M}':\sigma$ by a $\multimap$-normalization step. Then, for every $r \geq \overline{\mathtt{rk}}(\Pi)$:

$$\overline{\mathtt{w}}(\Pi', r) < \overline{\mathtt{w}}(\Pi, r)$$

Proof. Easy. ∎

So the desired results can be obtained.

Theorem 5.6 (Strong Polystep Soundness)
Let $\Pi \rhd \Gamma \vdash \mathtt{M}:\sigma$, and $\mathtt{M}$ β-reduces to $\mathtt{M}'$ in m steps. Then:

1. $m \leq |\mathtt{M}|^{\mathtt{d}(\Pi)+1}$
2. $|\mathtt{M}'| \leq |\mathtt{M}|^{\mathtt{d}(\Pi)+1}$

Proof.

1. By Lemma 5.3.2, Lemma 5.3.3 and by repeatedly using Lemma 5.5, since $|\mathtt{M}| \geq \mathtt{rk}(\Pi)$.
2. By repeatedly using Lemma 5.5 there is a derivation $\Sigma \rhd \Gamma \vdash \mathtt{M}':\sigma$ such that $\overline{\mathtt{w}}(\Sigma, r) < \overline{\mathtt{w}}(\Pi, r)$. By Lemma 5.3.3 $|\mathtt{M}'| = \overline{\mathtt{w}}(\Sigma, 1)$. Since clearly $\overline{\mathtt{w}}(\Sigma, 1) \leq \overline{\mathtt{w}}(\Sigma, r)$ we have $|\mathtt{M}'| < \overline{\mathtt{w}}(\Pi, r)$ and by Lemma 5.3.2 the conclusion follows. ∎

Theorem 5.7 (Polytime Soundness)
Let $\Pi \rhd \Gamma \vdash \mathtt{M}:\sigma$, then $\mathtt{M}$ can be evaluated to normal form on a Turing machine in time $O(|\mathtt{M}|^{3(\mathtt{d}(\Pi)+1)})$.

Proof. Clearly, as pointed in [21], a β reduction step $\mathtt{N} \to_\beta \mathtt{N}'$ can be simulated in time $O(|\mathtt{N}|^2)$ on a Turing machine. Let $\mathtt{M} \equiv \mathtt{M}_0 \to_\beta \mathtt{M}_1 \to_\beta \cdots \to_\beta \mathtt{M}_n$ be a reduction of $\mathtt{M}$ to normal form $\mathtt{M}_n$. By Theorem 5.6.2 $|\mathtt{M}_i| \leq |\mathtt{M}|^{\mathtt{d}(\Pi)+1}$ for $0 \leq i \leq n$, hence each step in the reduction takes time $O(|\mathtt{M}|^{2(\mathtt{d}(\Pi)+1)})$. Furthermore since by Theorem 5.6.1 n is $O(|\mathtt{M}|^{\mathtt{d}(\Pi)+1})$, the conclusion follows. ∎

Theorem 5.7 holds for every strategy. In fact analogously to [22] it could have been formulated as a strong polytime soundness, considering Turing machine with an oracle for strategies. We refer to [22] for further details.

5.2 Polynomial Time Completeness

In order to prove polynomial time completeness for STA we need to encode Turing machines (TM) configurations, transitions between such configurations and iterators. We encode input data, TM configurations and transitions following the lines of [18] and iterators as usual by Church numerals. In fact, in [18] the authors have shown that the multiplicative fragment of SLL is complete for PTIME. We here adapt their proof to show that it is also complete for FPTIME. Moreover, we exploit the fact that to a term can be assigned an infinite number of types, by introducing the notion of indexed types for typing the polynomial, and this allows us to skip the Lafont's programming discipline, i.e. the distinction between programs as terms typable by generic proofs and data as terms typable by homogeneous proofs.
Nevertheless, in order to make the bound in Theorem 5.6 polynomial, we show that each input data can be typed through derivations with fixed degree. In particular we show that they are typable with derivation with degree equal to 0.

5.2.1 Definability

We generalize the usual notion of lambda definability, given in [6], to different kinds of input data.

Definition 5.8
Let $f:\mathbb{I}_1\times\cdots\times\mathbb{I}_n\to\mathbb{O}$ be a total function and let elements $o\in\mathbb{O}$ and $i_j\in\mathbb{I}_j$, for $0\leq j\leq n$, be encoded by terms $\underline{o}$ and $\underline{i_j}$ such that $\Delta\vdash\underline{o}:\mathbf{O}$ and $\Delta_j\vdash\underline{i_j}:\mathbf{I}_j$ where $\Delta=\Delta_1,\ldots,\Delta_n$. Then, f is *definable* if, there exists a term $\underline{f}\in\Lambda$ such that $\Delta\vdash\underline{f}\,\underline{i_1}\cdots\underline{i_n}:\mathbf{O}$ and:

$$f i_1\cdots i_n=o\iff\underline{f}\,\underline{i_1}\cdots\underline{i_n}=_\beta\underline{o}$$

In what follows the symbol $\doteq$ denotes the definitional equivalence. Moreover, as usual $\mathtt{M}^n(\mathtt{N})$ denotes the term inductively defined as $\mathtt{M}^0(\mathtt{N})\doteq\mathtt{N}$ and $\mathtt{M}^{n+1}(\mathtt{N})\doteq\mathtt{M}(\mathtt{M}^n(\mathtt{N}))$ for every $n\in\mathbb{N}$.

5.2.2 Composition

The composition of two terms $\mathtt{M}$ and $\mathtt{N}$, denoted $\mathtt{M}\circ\mathtt{N}$ is definable as $\lambda\mathtt{z}.\mathtt{M}(\mathtt{N}\mathtt{z})$. In particular for composition we can derive the following rule:

$$\frac{\Gamma\vdash\mathtt{M}:A\multimap B\quad\Delta\vdash\mathtt{N}:\sigma\multimap A\quad\Gamma\#\Delta}{\Gamma,\Delta\vdash\mathtt{M}\circ\mathtt{N}:\sigma\multimap B}\ (comp)$$

Composition can be generalized to the n-ary case, denoted $\mathtt{M}_1\circ\mathtt{M}_2\circ\cdots\circ\mathtt{M}_n$, definable by the term $\lambda\mathtt{z}.\mathtt{M}_1(\mathtt{M}_2(\cdots(\mathtt{M}_n\mathtt{z})))$.

5.2.3 Multiplicative Unit

The *multiplicative unit* is definable by second order quantifier as:

$$\mathbf{1}\doteq\forall\alpha.\alpha\multimap\alpha$$

The constructors and destructors for this data type are definable as:

$$\mathtt{I}\doteq\lambda\mathtt{x}.\mathtt{x}\qquad\text{let }\mathtt{z}\text{ be }\mathtt{I}\text{ in }\mathtt{M}\doteq\mathtt{z}\mathtt{M}$$

The following are derived rules:

$$\frac{}{\vdash \mathtt{I}:\mathbf{1}} \qquad \frac{\Gamma\vdash \mathtt{N}:\mathbf{1} \quad \Delta\vdash \mathtt{M}:\sigma}{\Gamma,\Delta\vdash \text{let } \mathtt{N} \text{ be } \mathtt{I} \text{ in } \mathtt{M}:\sigma}$$

5.2.4 Restricted Weakening

STA is not an affine system, but we can introduce a restricted form of weakening. For doing so we adapt some results, presented in [18]. The following is a slight modification of Definition 1 in [18].

Definition 5.9
A type A is Π_1 if $\forall$-types occur only positively in it. A type A is $e\Pi_1$ if it contains positive occurrences of $\forall$-types or negative occurrences of inhabited $\forall$-types.

The following analogous of Theorem 1 of [18] will be used in the sequel. We refer to [18] for its proof.

Theorem 5.10
For any closed $e\Pi_1$ type A, there is a term $\mathtt{W}_A$ typable as $\vdash \mathtt{W}_A : A \multimap \mathbf{1}$.

5.2.5 Tensor product

Tensor product is definable by second order quantifier as

$$\sigma\otimes\tau \doteq \forall\alpha.(\sigma\multimap\tau\multimap\alpha)\multimap\alpha$$

The constructors and destructors for this data type are definable as:

$$\langle \mathtt{M},\mathtt{N}\rangle \doteq \lambda \mathtt{x}.\mathtt{x}\mathtt{M}\mathtt{N} \qquad \text{let } \mathtt{z} \text{ be } \mathtt{x},\mathtt{y} \text{ in } \mathtt{N} \doteq \mathtt{z}(\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{N})$$

The following are derived rules:

$$\frac{\Gamma\vdash \mathtt{M}:\sigma \quad \Delta\vdash \mathtt{N}:\tau \quad \Gamma\#\Delta}{\Gamma,\Delta\vdash \langle \mathtt{M},\mathtt{N}\rangle : \sigma\otimes\tau} \qquad \frac{\Gamma\vdash \mathtt{N}:\sigma\otimes\tau \quad \Delta,\mathtt{x}:\sigma,\mathtt{y}:\tau\vdash \mathtt{M}:\rho \quad \Gamma\#\Delta}{\Gamma,\Delta\vdash \text{let } \mathtt{N} \text{ be } \mathtt{x},\mathtt{y} \text{ in } \mathtt{M}:\rho}$$

We can use Theorem 5.10 to define projection functions. Let $\sigma_1\otimes\sigma_2$ be a closed $e\Pi_1$ type and $\mathtt{W}_{\sigma_1}$ and $\mathtt{W}_{\sigma_2}$ be the terms obtained applying Theorem 5.10. Then, we can define projection functions as follows:

$$\begin{array}{lcl} \pi_1^2 & \equiv & \lambda \mathtt{x}.\text{let } \mathtt{x} \text{ be } \mathtt{y}_1,\mathtt{y}_2 \text{ in } (\text{let } \mathtt{W}_{\sigma_2}\mathtt{y}_2 \text{ be } \mathtt{I} \text{ in } \mathtt{y}_1) \\ \pi_2^2 & \equiv & \lambda \mathtt{x}.\text{let } \mathtt{x} \text{ be } \mathtt{y}_1,\mathtt{y}_2 \text{ in } (\text{let } \mathtt{W}_{\sigma_1}\mathtt{y}_1 \text{ be } \mathtt{I} \text{ in } \mathtt{y}_2) \end{array}$$

n-ary tensor product can be easily defined through the binary one as follows:

$$\begin{array}{lcl} \sigma_1\otimes\cdots\otimes\sigma_n & \doteq & (\sigma_1\otimes\cdots\otimes\sigma_{n-1})\otimes\sigma_n \\ \langle \mathtt{M}_1,\ldots,\mathtt{M}_{n+1}\rangle & \doteq & \lambda \mathtt{x}.\mathtt{x}\langle \mathtt{M}_1,\ldots,\mathtt{M}_n\rangle \mathtt{M}_{n+1} \\ \text{let } \mathtt{z} \text{ be } \mathtt{x}_1,\vec{\mathtt{x}} \text{ in } \mathtt{M} & \doteq & \mathtt{z}(\lambda \mathtt{t}.\lambda \mathtt{x}_1.\text{let } \mathtt{t} \text{ be } \vec{\mathtt{x}} \text{ in } \mathtt{M}) \end{array}$$

In what follows σ^n denotes $\sigma\otimes\cdots\otimes\sigma$ n-times. Let $n\geq j\geq 2$ and $\sigma_1\otimes\cdots\otimes\sigma_n$ be a closed $e\Pi_1$ type, then we can define projections using the binary one as follows:

$$\pi_n^n = \pi_2^2 \qquad \pi_j^n \doteq \pi_j^{n-1}\circ\pi_1^2$$

5.2.6 Church numerals

To encode natural numbers we will use Church numerals:

$$\underline{\mathtt{n}} \doteq \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}^n(\mathtt{z})$$

Church numerals are typable by the usual SLL type for natural numbers

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

nevertheless we prefer, for reasons that will be explained later, to introduce the following more general notion of indexed type.

Definition 5.11
The *indexed type* $\mathbf{N}_i$ for each $i \in \mathbb{N}$ is defined as:

$$\mathbf{N}_i \doteq \forall \alpha.!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$$

where $!^i$ stands for $\underbrace{!...!}_{i}$.

Clearly $\mathbf{N}_1 \equiv \mathbf{N}$. The following lemma holds.

Lemma 5.12
For each Church numerals $\underline{\mathtt{n}}$ and for each $i > 0 \in \mathbb{N}$:

$$\vdash \underline{\mathtt{n}} : \mathbf{N}_i$$

Proof. By induction on n. ∎

5.2.7 Iteration

As usual Church numerals behave as iterators. In fact we can define the *iteration* $\mathtt{N}$ times of the term $\mathtt{S}$ (representing the *step* function) over the term $\mathtt{B}$ (representing the *base* function) as:

$$\mathtt{Iter}(\mathtt{N},\mathtt{S},\mathtt{B}) \doteq \mathtt{NSB}$$

Clearly for every Church numeral $\underline{\mathtt{n}}$:

$$\mathtt{Iter}(\underline{\mathtt{n}},\mathtt{S},\mathtt{B}) \to_\beta^* \mathtt{S}^n\mathtt{B}$$

moreover it is easy to verify that the following is a derived rule:

$$\frac{\Gamma \vdash \mathtt{N} : \mathbf{N} \quad \Delta_1 \vdash \mathtt{B} : A \quad \Delta_2 \vdash \mathtt{S} : A \multimap A}{\Gamma, \Delta_1, !\Delta_2 \vdash \mathtt{Iter}(\mathtt{N},\mathtt{S},\mathtt{B}) : A}$$

It is worth noting that the step function is iterable only if it is definable through a term typable with type $A \multimap A$ for some linear type A. This in contrast with what happens in linear logic, where in general step functions proving $!A \multimap A$ are allowed.

5.2.8 Polynomials

Successor, addition and multiplication are definable through the usual terms [6].

LEMMA 5.13
The terms $\underline{\mathtt{succ}} \doteq \lambda p.\lambda s.\lambda z.s(psz)$, $\underline{\mathtt{add}} \doteq \lambda p.\lambda q.\lambda s.\lambda z.ps(qsz)$ and $\underline{\mathtt{mul}} \doteq \lambda p.\lambda q.\lambda s.p(qs)$ define respectively the successor, addition and multiplication functions. They are typable in STA for $i, j > 0 \in \mathbb{N}$ as:

$$\begin{array}{ll} \vdash_N \underline{\mathtt{succ}} & : \mathbf{N}_i \multimap \mathbf{N}_{i+1} \\ \vdash_N \underline{\mathtt{add}} & : \mathbf{N}_i \multimap \mathbf{N}_j \multimap \mathbf{N}_{\max(i,j)+1} \\ \vdash_N \underline{\mathtt{mul}} & : \mathbf{N}_j \multimap !^j\mathbf{N}_i \multimap \mathbf{N}_{i+j} \end{array}$$

PROOF. Easy. ∎

Note that the terms $\underline{\mathtt{succ}}, \underline{\mathtt{add}}$ and $\underline{\mathtt{mul}}$ cannot be typed by the usual types $\mathbf{N} \multimap \mathbf{N}$ (for the first) and $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$ (for the last two). As consequence they cannot be iterated through Church numerals iteration, but they can be composed to obtain all the polynomials.

EXAMPLE 5.14
If we want to multiply two natural numbers we can use $\underline{\mathtt{mul}}$ typed as:

$$\vdash_N \underline{\mathtt{mul}} : \mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2$$

If we want to multiply three natural numbers, we can use the term $\lambda x.\lambda y.\lambda z.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y)z$. Such term is typable by the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi \rhd x:\mathbf{N}, y:!\mathbf{N} \vdash_N \underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y) :!!\mathbf{N} \multimap \mathbf{N}_3 \quad \dfrac{\dfrac{z:\mathbf{N} \vdash_N z:\mathbf{N}}{z:!!\mathbf{N} \vdash_N z:!!\mathbf{N}}}{}}{x:\mathbf{N}, y:!\mathbf{N}, z:!!\mathbf{N} \vdash_N \underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y)z : \mathbf{N}_3}}{x:\mathbf{N}, y:!\mathbf{N} \vdash_N \lambda z.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y)z :!!\mathbf{N} \multimap \mathbf{N}_3}}{x:\mathbf{N} \vdash_N \lambda y.\lambda z.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y)z :!\mathbf{N} \multimap !!\mathbf{N} \multimap \mathbf{N}_3}}{\vdash_N \lambda x.\lambda y.\lambda z.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y)z : \mathbf{N} \multimap !\mathbf{N} \multimap !!\mathbf{N} \multimap \mathbf{N}_3}$$

where Π is the following derivation:

$$\dfrac{\dfrac{\vdash_N \underline{\mathtt{mul}} : \mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2 \quad x:\mathbf{N} \vdash_N x:\mathbf{N}}{x:\mathbf{N} \vdash_N \underline{\mathtt{mul}}\, x :!\mathbf{N} \multimap \mathbf{N}_2} \quad \dfrac{y:\mathbf{N} \vdash_N y:\mathbf{N}}{y:!\mathbf{N} \vdash_N y:!\mathbf{N}}}{\vdash_N \underline{\mathtt{mul}} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3 \qquad x:\mathbf{N}, y:!\mathbf{N} \vdash_N (\underline{\mathtt{mul}}\, x\, y) : \mathbf{N}_2}$$
$$x:\mathbf{N}, y:!\mathbf{N} \vdash_N \underline{\mathtt{mul}}(\underline{\mathtt{mul}}\, x\, y) :!!\mathbf{N} \multimap \mathbf{N}_3$$

So in particular the two occurrences of $\underline{\mathtt{mul}}$ need to be typed by different types. In the derivation we have assigned the type $\mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2$ to the innermost and $\mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3$ to the outermost one.

It is important to stress that the change of type as in the Example 5.14, in particular on the number of modalities, does not depend on the values of data.
We can finally show that STA is complete for polynomials.

THEOREM 5.15 (Polynomial Completeness)
Let P be a polynomial in the variable X and $deg(P)$ be its degree. Then there is a term $\underline{\mathtt{P}}$ defining P, typable as :

$$\mathtt{x}:!^{deg(P)}\mathbf{N}\vdash_{\mathrm{N}}\underline{\mathtt{P}}:\mathbf{N}_{2deg(P)+1}$$

PROOF. Consider P in Horner normal form, i.e., $P=a_0+X(a_1+X(\cdots(a_{n-1}+Xa_n)\cdots)$. By induction on $deg(P)$ we show something stronger, i.e., for $i>0$, it is derivable:

$$\mathtt{x}_0:\mathbf{N}_i,\mathtt{x}_1:!^i\mathbf{N}_i,\ldots,\mathtt{x}_n:!^{i(deg(P^*)-1)}\mathbf{N}_i\vdash_{\mathrm{N}}\underline{\mathtt{P}^*}:\mathbf{N}_{i(deg(P^*))+deg(P^*)+1}$$

where $P^*=a_0+X_0(a_1+X_1(\cdots(a_{n-1}+X_n\,a_n)\cdots)$ so the conclusion follows using the (m) rule and taking $i=1$.
Base case is trivial, so consider $P^*=a_0+X_0(P')$. By induction hypothesis:

$$\mathtt{x}_1:\mathbf{N}_i,\ldots,\mathtt{x}_n:!^{i(deg(P')-1)}\mathbf{N}_i\vdash_{\mathrm{N}}\underline{\mathtt{P}'}:\mathbf{N}_{i(deg(P'))+deg(P')+1}$$

Take $\underline{\mathtt{P}^*}\equiv\underline{\mathtt{add}}(\underline{a_0},\underline{\mathtt{mul}}(\mathtt{x}_0,\underline{\mathtt{P}'}))$, clearly we have:

$$\mathtt{x}_0:\mathbf{N}_i,\mathtt{x}_1:!^i\mathbf{N}_i,\ldots,\mathtt{x}_n:!^{i(deg(P')-1)+i}\mathbf{N}_i\vdash_{\mathrm{N}}\underline{\mathtt{P}^*}:\mathbf{N}_{i(deg(P')+1)+deg(P')+1+1}$$

Since $deg(P^*)=deg(P')+1$: it follows

$$\mathtt{x}_0:\mathbf{N}_i,\mathtt{x}_1:!^i\mathbf{N}_i,\ldots,\mathtt{x}_n:!^{i(deg(P^*)-1)}\mathbf{N}_i\vdash_{\mathrm{N}}\underline{\mathtt{P}^*}:\mathbf{N}_{i(deg(P^*))+deg(P^*)+1}$$

Now by taking $i=1$ and repeatedly applying (m) rule we conclude

$$\mathtt{x}:!^{deg(P)}\mathbf{N}\vdash_{\mathrm{N}}\underline{\mathtt{P}}\equiv\underline{\mathtt{P}^*}[\mathtt{x}/\mathtt{x}_1,\cdots,\mathtt{x}/\mathtt{x}_n]:\mathbf{N}_{2deg(P)+1}$$ ∎

5.2.9 Booleans

Let us encode booleans, as in [18], by:

$$\mathbf{0}\doteq\lambda\mathtt{x}.\lambda\mathtt{y}.\langle\mathtt{x},\mathtt{y}\rangle\qquad\mathbf{1}\doteq\lambda\mathtt{x}.\lambda\mathtt{y}.\langle\mathtt{y},\mathtt{x}\rangle$$

By convention we use **0** and **1** for *true* and *false* respectively. They can be typed in STA by the following $e\Pi_1$ type:

$$\mathbf{B}\doteq\forall\alpha.\alpha\multimap\alpha\multimap(\alpha\otimes\alpha)$$

It is easy to verify that the following are derived rules:

$$\frac{}{\vdash\mathbf{0}:\mathbf{B}}\ (\mathbf{B}_0R)\qquad\frac{}{\vdash\mathbf{1}:\mathbf{B}}\ (\mathbf{B}_1R)$$

From the fact that **B** is a $e\Pi_1$ type we have the terms:

$$\begin{array}{rcl}\underline{\mathtt{Or}}&\doteq&\lambda\mathtt{b}_1.\lambda\mathtt{b}_2.\pi^2_1(\mathtt{b}_1\mathbf{0}\mathtt{b}_2)\\ \underline{\mathtt{And}}&\doteq&\lambda\mathtt{b}_1.\lambda\mathtt{b}_2.\pi^2_1(\mathtt{b}_1\mathtt{b}_2\mathbf{1})\\ \underline{\mathtt{Not}}&\doteq&\lambda\mathtt{b}_1.\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{b}_1\mathtt{y}\mathtt{x}\end{array}$$

defining the boolean disjunction, conjunction and negation respectively, which are typable as:

$$\vdash \underline{\mathtt{And}} : \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B} \qquad \vdash \underline{\mathtt{Or}} : \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B} \qquad \vdash \underline{\mathtt{Not}} : \mathbf{B} \multimap \mathbf{B}$$

Moreover, we have terms acting as weakening and contraction on booleans:

$$\begin{array}{lcl} \mathtt{W}_{\mathbf{B}} & \equiv & \lambda \mathtt{z}.\mathtt{let\ z\,II\ be\ x,y\ in\ (let\ y\ be\ I\ in\ x)} \\ \mathtt{C}_{\mathbf{B}} & \equiv & \lambda \mathtt{z}.\pi_1^2(\mathtt{z}\langle \mathbf{0},\mathbf{0}\rangle\langle \mathbf{1},\mathbf{1}\rangle) \end{array}$$

which are respectively typable as:

$$\vdash \mathtt{W}_{\mathbf{B}} : \mathbf{B} \multimap \mathbf{1} \qquad\qquad \vdash \mathtt{C}_{\mathbf{B}} : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B}$$

The above functions are useful to prove the following lemma.

Lemma 5.16
Each boolean total function $f : \mathbb{B}^n \to \mathbb{B}^m$, where $n, m \geq 1$, can be defined by a term $\underline{\mathtt{f}}$ typable in STA as $\vdash \underline{\mathtt{f}} : \mathbf{B}^n \multimap \mathbf{B}^m$.

Proof. Easy. ∎

In the sequel it will be usefull the following analogous of Lemma 1 of [18]:

Lemma 5.17
Let A be a (not necessarily closed) $e\Pi_1$ type. Then the following is a derived rule:

$$\frac{\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \mathbf{B} \quad \vdash_{\mathrm{N}} \mathtt{L} : A \quad \vdash_{\mathrm{N}} \mathtt{R} : A}{\Gamma \vdash_{\mathrm{N}} \mathtt{if\ M\ then\ L\ else\ R} : A}\ (\mathbf{B}E)$$

5.2.10 Strings

Strings of booleans can be encoded as:

$$[\,] \doteq \lambda \mathtt{c}.\lambda \mathtt{z}.\mathtt{z} \qquad [\mathtt{b}_0, \mathtt{b}_1, \ldots, \mathtt{b}_n] \doteq \lambda \mathtt{c}.\lambda \mathtt{z}.\mathtt{c}\mathtt{b}_0(\cdots(\mathtt{c}\mathtt{b}_n\mathtt{z})\cdots)$$

where $\mathtt{b}_i \in \{\mathbf{0}, \mathbf{1}\}$. Boolean strings are typable in STA by the indexed type:

$$\mathbf{S}_i \doteq \forall \alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

In particular for each $n, i > 0 \in \mathbb{N}$ the following is a derived rule:

$$\frac{}{\mathtt{b}_0 : \mathbf{B}, \ldots, \mathtt{b}_n : \mathbf{B} \vdash [\mathtt{b}_0, \ldots, \mathtt{b}_n] : \mathbf{S}_i}\ (\mathbf{S}_i R)$$

In the sequel we will use the following.

Lemma 5.18
The term $\mathbf{len} \doteq \lambda \mathtt{c}.\lambda \mathtt{s}.\mathtt{c}(\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{let}\ \mathtt{W}_{\mathbf{B}}\mathtt{x}\ \mathtt{be\ I\ in\ sy})$ defines the function returning the length of an input string. It is typable in STA with typing:

$$\vdash \mathbf{len} : \mathbf{S}_i \multimap \mathbf{N}_i$$

Proof. Easy. ∎

5.2.11 Turing Machine

We define Turing machine configurations by terms of the shape

$$\lambda c.\langle \mathtt{cb}_0^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{Q}\rangle$$

where $\mathtt{cb}_0^l \circ \cdots \circ \mathtt{cb}_n^l$ and $\mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r$ represent respectively the left and the right part of the tape, while $\mathtt{Q} \equiv \langle \mathtt{b}_1, \cdots, \mathtt{b}_n\rangle$ is a n-ary tensor product of boolean values representing the current state.
We assume, without loss of generality, that by convention the left part of the tape is represented in a reversed order, that the alphabet is composed by the two symbols **0** and **1**, that the scanned symbol is the first symbol in the right part and that final states are divided in accepting and rejecting.

Definition 5.19
The *indexed type* $\mathbf{TM}_i^k$ for each $i, k \in \mathbb{N}$ is defined as:

$$\mathbf{TM}_i^k \doteq \forall\alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha)^2 \otimes \mathbf{B}^k)$$

The above indexed type is useful to type Turing machine configurations.

Lemma 5.20
Let $k \in \mathbb{N}$. Every term $\lambda c.\langle \mathtt{cb}_0^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r, \langle \mathtt{q}_0, \ldots, \mathtt{q}_k\rangle\rangle$ defines a Turing machine configuration. For every $i > 0$ such terms are typable in STA as:

$$\vdash \lambda c.\langle \mathtt{cb}_0^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r, \langle \mathtt{q}_0, \ldots, \mathtt{q}_k\rangle\rangle : \mathbf{TM}_i^k$$

The initial configuration of a Turing machine is represented by a tape of fixed length filled by **0** with the head at the begin of the tape and in the initial state Q_0. The following lemma shows how the initial configuration of a Turing machine can be obtained starting from a numeral representing the length of the tape.

Lemma 5.21
The term $\mathbf{Init} \doteq \lambda t.\lambda c.\langle \lambda z.z, \lambda z.t(c\mathbf{0})z, \underline{\mathtt{Q}_0}\rangle$ defines the function that, taking as input a Church numeral $\underline{\mathtt{n}}$, gives as output a Turing machine with tape of length n filled by **0**'s in the initial state $\mathtt{Q}_0 \equiv \langle \mathtt{q}_0, \ldots, \mathtt{q}_k\rangle$ and with the head at the beginning of the tape. For each $i \in \mathbf{N}$ it is typable in STA as:

$$\vdash \mathbf{Init} : \mathbf{N}_i \multimap \mathbf{TM}_i^k$$

Proof. Easy. ∎

Following [18], in order to show that Turing machine transitions are definable we consider two distinct phases. In the first one the TM configuration is decomposed to extract the first symbol of each part of the tape. In the second phase the symbols obtained in the previous one are combined, depending on the transition function, to reconstruct the tape after the transition step. In order to type the decomposition of a TM configuration we will use the type $\mathbf{ID}_i^k$ defined as:

$$\mathbf{ID}_i^k \doteq \forall\alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha)^2 \otimes (\mathbf{B} \multimap \alpha \multimap \alpha) \otimes \mathbf{B} \otimes (\mathbf{B} \multimap \alpha \multimap \alpha) \otimes \mathbf{B} \otimes \mathbf{B}^k)$$

The decomposition phase is described in the following lemma.

LEMMA 5.22
The term:

$$\mathbf{Dec} \doteq \lambda \mathtt{s}.\lambda \mathtt{c}.\mathtt{let}\ \mathtt{s}(\mathtt{F}[\mathtt{c}])\ \mathtt{be}\ \mathtt{l},\mathtt{r},\mathtt{q}\ \mathtt{in}\ \ \mathtt{let}\ \mathtt{l}\langle \mathtt{I}, \lambda \mathtt{x}.\mathtt{let}\ \mathtt{W_B x}\ \mathtt{be}\ \mathtt{I}\ \mathtt{in}\ \mathtt{I}, \mathbf{0}\rangle \mathtt{be}\ \mathtt{t}_l, \mathtt{c}_l, \mathtt{b}_0^l\ \mathtt{in}$$
$$\mathtt{let}\ \mathtt{r}\langle \mathtt{I}, \lambda \mathtt{x}.\mathtt{let}\ \mathtt{W_B x}\ \mathtt{be}\ \mathtt{I}\ \mathtt{in}\ \mathtt{I}, \mathbf{0}\rangle\ \mathtt{be}\ \mathtt{t}_r, \mathtt{c}_r, \mathtt{b}_0^r\ \mathtt{in}\ \langle \mathtt{t}_l, \mathtt{t}_r, \mathtt{c}_l, \mathtt{b}_0^l, \mathtt{c}_r, \mathtt{b}_0^r, \mathtt{q}\rangle$$

where $\mathtt{F}[\mathtt{c}] \doteq \lambda \mathtt{b}.\lambda \mathtt{z}.\mathtt{let}\ \mathtt{z}\ \mathtt{be}\ \mathtt{g},\mathtt{h},\mathtt{i}\ \mathtt{in}\ \langle \mathtt{hi} \circ \mathtt{g}, \mathtt{c}, \mathtt{b}\rangle$ is typable as:

$$\vdash \mathbf{Dec} : \mathbf{TM}_i^k \multimap \mathbf{ID}_i^k$$

Its behavior is to decompose a configuration as:

$$\mathbf{Dec}(\lambda \mathtt{c}.\langle \mathtt{cb}_0^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{Q}\rangle) \to_\beta^*$$
$$\lambda \mathtt{c}.\langle \mathtt{cb}_1^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_1^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{c}, \mathtt{b}_0^l, \mathtt{c}, \mathtt{b}_0^r, \mathtt{Q}\rangle$$

PROOF. To verify that **Dec** has the above typing and the intended behavior is boring but easy. ∎

Analogously for the combining phase we have the following lemma.

LEMMA 5.23
The term:

$$\mathbf{Com} \doteq \lambda \mathtt{s}.\lambda \mathtt{c}.\mathtt{let}\ \mathtt{sc}\ \mathtt{be}\ \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r,\mathtt{b}_r,\mathtt{q}\ \mathtt{in}$$
$$\mathtt{let}\ \underline{\delta}\langle \mathtt{b}_r, \mathtt{q}\rangle\ \mathtt{be}\ \mathtt{b}', \mathtt{q}', \mathtt{m}\ \mathtt{in}\ (\mathtt{if}\ \mathtt{m}\ \mathtt{then}\ \mathtt{R}\ \mathtt{else}\ \mathtt{L})\mathtt{b}'\mathtt{q}'\langle \mathtt{l}, \mathtt{r}, \mathtt{c}_l, \mathtt{b}_l, \mathtt{c}_r\rangle$$

where

$$\begin{array}{rcl} \mathtt{R} & \doteq & \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s}.\mathtt{let}\ \mathtt{s}\ \mathtt{be}\ \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r\ \mathtt{in}\ \langle \mathtt{c}_r\mathtt{b}' \circ \mathtt{c}_l\mathtt{b}_l \circ \mathtt{l}, \mathtt{r}, \mathtt{q}'\rangle \\ \mathtt{L} & \doteq & \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s}.\mathtt{let}\ \mathtt{s}\ \mathtt{be}\ \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r\ \mathtt{in}\ \langle \mathtt{l}, \mathtt{c}_l\mathtt{b}_l \circ \mathtt{c}_r\mathtt{b}' \circ \mathtt{r}, \mathtt{q}'\rangle \end{array}$$

is typable as:

$$\vdash_{\mathrm{N}} \mathbf{Com} : \mathbf{ID}_i^k \multimap \mathbf{TM}_i^k$$

Its behavior is, depending on the δ transition function, to combine the symbols returning a configuration as:

$$\begin{array}{l} \mathbf{Com}\ (\lambda \mathtt{c}.\langle \mathtt{cb}_1^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_1^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{c}, \mathtt{b}_0^l, \mathtt{c}, \mathtt{b}_0^r, \mathtt{Q}\rangle) \\ \to_\beta^* \lambda \mathtt{c}.\langle \mathtt{cb}' \circ \mathtt{cb}_0^l \circ \mathtt{cb}_1^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_1^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{Q}'\rangle \quad \text{if } \delta(\mathtt{b}_0^r, \mathtt{Q}) = (\mathtt{b}', \mathtt{Q}', \mathrm{Right}) \\ \text{or} \\ \to_\beta^* \lambda \mathtt{c}.\langle \mathtt{cb}_1^l \circ \cdots \circ \mathtt{cb}_n^l, \mathtt{cb}_0^l \circ \mathtt{cb}' \circ \mathtt{cb}_1^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{Q}'\rangle \quad \text{if } \delta(\mathtt{b}_0^r, \mathtt{Q}) = (\mathtt{b}', \mathtt{Q}', \mathrm{Left}) \end{array}$$

PROOF. To verify that **Com** has the above typing and the intended behavior is boring but easy. ∎

By combining the above terms we obtain an entire Turing machine transition step.

LEMMA 5.24
The term $\mathbf{Tr} \doteq \mathbf{Com} \circ \mathbf{Dec}$ defines a Turing machine transition step and is typable as:

$$\vdash_{\mathrm{N}} \mathbf{Tr} : \mathbf{TM}_i^k \multimap \mathbf{TM}_i^k$$

PROOF. Easy, by Lemma 5.22 and Lemma 5.23. ∎

We need a term that initializes a Turing machine with an input string.

LEMMA 5.25
The term

$$\mathbf{In} \doteq \lambda \mathtt{s}.\lambda \mathtt{m}.\mathtt{s}(\lambda \mathtt{b}.(\mathtt{T}\mathtt{b}) \circ \mathbf{Dec})\mathtt{m}$$

where

$$\mathtt{T} \doteq \lambda \mathtt{b}.\lambda \mathtt{s}.\lambda \mathtt{c}.\text{let } \mathtt{sc} \text{ be } \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r,\mathtt{b}_r,\mathtt{q} \text{ in let } \mathtt{W}_{\mathbf{B}}\mathtt{b}_r \text{ be } \mathtt{I} \text{ in } \mathtt{R}\mathtt{b}\mathtt{q}\langle \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r\rangle$$

and $\mathtt{R} \doteq \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s}.\text{let } \mathtt{s} \text{ be } \mathtt{l},\mathtt{r},\mathtt{c}_l,\mathtt{b}_l,\mathtt{c}_r \text{ in } \langle \mathtt{c}_r\mathtt{b}' \circ \mathtt{c}_l\mathtt{b}_l \circ \mathtt{l},\mathtt{r},\mathtt{q}'\rangle$, defines the function that, when supplied by a boolean string and a Turing machine, writes the input string on the tape of the Turing machine. Such a term is typable as

$$\vdash_{\mathrm{N}} \mathbf{In} : \mathbf{S} \multimap \mathbf{TM}_i^k \multimap \mathbf{TM}_i^k$$

Finally we need a term that returns the acceptance or not of a final configuration.

LEMMA 5.26
Let f be a function deciding if a state is accepting or rejecting. Then the term:

$$\mathbf{Ext} \doteq \lambda \mathtt{s}.\text{let } \mathtt{s}(\lambda \mathtt{b}.\lambda \mathtt{c}.\text{let } \mathtt{W}_{\mathbf{B}}\mathtt{b} \text{ be } \mathtt{I} \text{ in } \mathtt{c}) \text{ be } \mathtt{l},\mathtt{r},\mathtt{q} \text{ in } (\mathtt{l} \circ \mathtt{r})(\underline{\mathtt{f}}\mathtt{q})$$

defines the function that given a Turing machine configuration returns **0** if it is accepting, **1** otherwise. It is typable in STA as:

$$\vdash_{\mathrm{N}} \mathbf{Ext} : \mathbf{TM}_i^k \multimap \mathbf{B}$$

PROOF. Easy. Note that the existence of the term $\underline{\mathtt{f}}$ is assured by Lemma 5.16. ∎

Now we can show that STA is complete for PTIME.

THEOREM 5.27 (PTIME Completeness)
Let a decision problem $\mathcal{P}$ be decided in *polynomial time* P, where $deg(P) = m$, and in *polynomial space* Q, where $deg(Q) = l$, by a Turing machine $\mathcal{M}$. Then it is definable by a term $\underline{\mathtt{M}}$ typable in STA as:

$$\mathtt{s} :!^{max(l,m,1)+1}\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{M}} : \mathbf{B}$$

PROOF. By Theorem 5.15: $\mathtt{s}_p :!^m\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{P}}[\mathbf{len}\mathtt{s}_p/\mathtt{x}] : \mathbf{N}_{2m+1}$ and $\mathtt{s}_q :!^l\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{Q}}[\mathbf{len}\mathtt{s}_q/x] : \mathbf{N}_{2l+1}$. Furthermore, by composition:

$$\mathtt{s} : \mathbf{S}, \mathtt{q} : \mathbf{N}_{2l+1}, \mathtt{p} : \mathbf{N}_{2m+1} \vdash_{\mathrm{N}} \mathbf{Ext}(\mathtt{p}\mathbf{Tr}(\mathbf{In}s(\mathbf{Init}(\mathtt{q})))) : \mathbf{B}$$

so by substitution and by some applications of (m) rule the conclusion follows. ∎

Without loss of generality we can assume that a Turing machine stops in a final configuration with the head at the beginning of the tape. This means that the final configuration is represented by terms of the shape:

$$\lambda \mathtt{c}.\langle \lambda \mathtt{z}.\mathtt{z}, \mathtt{cb}_0^r \circ \cdots \circ \mathtt{cb}_m^r, \mathtt{Q} \rangle$$

The following lemma shows that the function extracting the output string from the final configuration is easily definable.

Lemma 5.28
The term:

$$\mathbf{Ext_F} \doteq \lambda \mathtt{s}.\lambda \mathtt{c}.\mathtt{let}\ \mathtt{sc}\ \mathtt{be}\ \mathtt{l},\mathtt{r},\mathtt{q}\ \mathtt{in}\ \mathtt{let}\ \mathtt{W}_{\mathbf{B}^k}\mathtt{q}\ \mathtt{be}\ \mathtt{I}\ \mathtt{in}\ \mathtt{l} \circ \mathtt{r}$$

defines the function that given an accepting Turing machine configuration extracts the result from the tape. It is typable in STA as:

$$\vdash_{\mathrm{N}} \mathbf{Ext_F} : \mathbf{TM}_i^k \multimap \mathbf{S}_i$$

Proof. Easy. ∎

So we can conclude that STA is also complete for FPTIME.

Theorem 5.29 (FPTIME Completeness)
Let a function $\mathcal{F}$ be computed in *polynomial time* P, where $deg(P) = m$, and in *polynomial space* Q, where $deg(Q) = l$, by a Turing machine $\mathcal{M}$. Then it is definable by a term $\underline{\mathtt{M}}$ typable in STA as:

$$!^{max(l,m,1)+1}\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{M}} : \mathbf{S}_{2l+1}$$

Proof. Similar to the proof of Theorem 5.27 but using $\mathbf{Ext_F}$. By Theorem 5.15: $\mathtt{s}_p : !^m\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{P}}[\mathbf{len}\mathtt{s}_p/\mathtt{x}] : \mathbf{N}_{2m+1}$ and $\mathtt{s}_q : !^l\mathbf{S} \vdash_{\mathrm{N}} \underline{\mathtt{Q}}[\mathbf{len}\mathtt{s}_q/x] : \mathbf{N}_{2l+1}$. By composition:

$$\mathtt{s} : \mathbf{S}, \mathtt{q} : \mathbf{N}_{2l+1}, \mathtt{p} : \mathbf{N}_{2m+1} \vdash_{\mathrm{N}} \mathbf{Ext_F}(\mathtt{p}\mathbf{Tr}(\mathbf{In}s(\mathbf{Init}(\mathtt{q})))) : \mathbf{S}_{2l+1}$$

so by substitution and some applications of (m) rule the conclusion follows. ∎

6 Conclusion and future work

We have shown a type assignment system for λ-calculus, which is correct and complete for polynomial time, i.e., each term that can be typed can be reduced to normal form in time polynomial with respect to its size, and all polynomial functions can be computed by a well typed term. The next step is to consider the problem of type inference. We conjecture because of the presence of the second order quantifier that this problem is undecidable. It would be possible to follow the same method as in [23] for System F.
In an incoming paper [13], we proved that, if we restrict ourselves to consider just the propositional fragment, the type inference problem for STA is decidable in polynomial time in the length of the input term. Moreover, for the whole system, we showed an algorithm generating all the constraints that need to be satisfied in order to type a given term. This

algorithm can be used for checking the typability in some particular cases. We leave for future investigations the checking of our conjecture and, in case this will be proved true, the design of an "approximate" typability algorithm, checking for typability with respect to given constraints on the shape of the possible typings.

References

[1] Andrea Asperti. Light affine logic. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science* (LICS '98), pages 300–308. IEEE Computer Society, 1998.

[2] Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.

[3] Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: a language for polynomial time computation. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures* (FoSSaCS '04), volume 2987 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[4] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science* (LICS '04), pages 266–275. IEEE Computer Society, 2004.

[5] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1):41–62, 2009.

[6] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier/North-Holland, Amsterdam, London, New York, revised edition, 1984.

[7] Paolo Coppola, Ugo Dal Lago, and Simona Ronchi Della Rocca. Elementary affine logic and the call by value lambda calculus. In *Proceedings of the 8th International Conference on Typed Lambda-Calculi and Applications* (TLCA '05), volume 3461 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2005.

[8] Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003.

[9] Marco Gaboardi. *Linearity: an Analytic Tool in the study of Complexity and Semantics of Programming Languages*. PhD thesis, Università degli Studi di Torino - Institut National Polytechnique de Lorraine, 2007.

[10] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of PSPACE. In *35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL 2008, San Francisco, January 10-12, 2008, Proceedings*, pages 121–131, 2008.

[11] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. Soft linear logic and polynomial complexity classes. In *Proceedings of the Second Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2007)*, volume 205 of *Electronic Notes in Theoretical Computer Science*, pages 67–87. Elsevier, 2008.

[12] Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for λ-calculus. In *Proceedings of the 21st International Workshop on Computer Science Logic* (CSL '07), volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.

[13] Marco Gaboardi and Simona Ronchi Della Rocca. Type inference for a polynomial lambda-calculus. In *Types for proofs and programs 2008, TYPES'08*, Lecture Notes in Computer Science. Springer, 2009. Accepted.

[14] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
[15] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
[16] Jean Goubault-Larrecq and Ian Mackie. *Proof Theory and Automated Deduction*, volume 6 of *Applied Logic Series*. Kluwer Academic Publishers, Dordrecht, 1997.
[17] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.
[18] Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In *ICTCS*, volume 2841 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2003.
[19] Simona Ronchi Della Rocca and Luca Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3), 1997.
[20] Morten H. Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.
[21] Kazushige Terui. Light affine lambda calculus and polytime strong normalization. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science* (LICS '01), pages 209–220. IEEE Computer Society, 2001.
[22] Kazushige Terui. *Light logic and polynomial time computation*. PhD thesis, Keio University, 2002.
[23] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.

Received May 11, 2008