Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 205 (2008) 67–87
www.elsevier.com/locate/entcs

Soft Linear Logic and Polynomial Complexity Classes

Marco Gaboardi[1]

Dipartimento di Informatica, Università degli studi di Torino
Corso Svizzera 185, 10149 Torino, Italy

Jean-Yves Marion[2]

Nancy-University, ENSMN-INPL, Loria
B.P. 239, 54506 Vandoeuvre-lès-Nancy, France

Simona Ronchi Della Rocca[3]

Dipartimento di Informatica, Università degli studi di Torino
Corso Svizzera 185, 10149 Torino, Italy

Abstract

We describe some results inspired to Lafont's Soft Linear Logic (SLL) which is a subsystem of second-order linear logic with restricted rules for exponentials, correct and complete for polynomial time computations. SLL is the basis for the design of type assignment systems for lambda-calculus, characterizing the complexity classes PTIME, PSPACE and NPTIME. PTIME is characterized by a type assignments system where types are a proper subset of SLL formulae. The characterization consists in the fact that a well typed term can be reduced to normal form by a number of beta-reductions polynomial in its lenght, and moreover all polynomial time functions can be computed by well typed terms. PSPACE is characterized by a type assignment system obtained from the previous one, by extending the set of types by a type for booleans, and the lambda-calculus by two boolean constants and a conditional constructor. The system assigns types to terms in such a way that the evaluation of programs (closed terms of type boolean) can be performed carefully in polynomial space. Moreover all polynomial space decision problems can be computed by terms typable in this system. In order to characterize NPTIME we extend the lambda-calculus by a nondeterministic choice operator, and the system by a rule for dealing with this new term constructor.

Keywords: Implicit computational complexity, linear logic, polynomial space, type assignment.

1 Introduction

A recent research field in Theoretical Computer Science is the design of programming languages with bounded computational complexity. In fact, guaranteeing and

[1] Email: gaboardi@di.unito.it
[2] Email: Jean-Yves.Marion@loria.fr
[3] Email: ronchi@di.unito.it

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2008.03.066

certifying a limited resources usage is of central importance for various aspects of computer science. A typical example is the one of a network constituted of small and mobile devices with bounded computational resources that receive programs to be executed from the network itself.

One of the more promising approaches to the design of programming languages with bounded complexity is based on the use of λ-calculus as paradigmatic programming language, and on the design of type assignment systems for λ-terms, where types guarantee, besides the functional correctness, also the desired complexity bound. So the complexity can be checked statically at compilation time, in ML style.

Useful tools for this aim are the so called light logics, derived from Girard's Linear Logic [3], where cut elimination is bounded in time by the size of the proof. In these logics a complete characterization of different complexity classes has been given, considering proofs as programs and the cut-elimination as computational mechanism. The idea is to exploit the Curry-Howard isomorphism, and so to use formulae of light logics as types for (extensions of) the λ-calculus, in such a way that well typed terms inherit the good properties of their types, with respect to the complexity bound.

Here we will describe some results inspired by Lafont's Soft Linear Logic (SLL) [7], which is a subsystem of second-order linear logic with restricted rules for exponentials, correct and complete for polynomial time computations. SLL has been the starting point for the characterization of PTIME and PSPACE through type assignment systems, presented respectively in [5] and [4]. Here we use a similar approach to characterize the complexity class NPTIME. But, in order to give a complete description of our methodology, in this paper we will present all the three results, in an incremental way. First we will present STA [5], a type assignments system for λ-calculus, where types are a proper subset of SLL formulae. This restriction is necessary to obtain the property of subject reduction, which is essential for making statically both type assignment and type checking. A term which can be typed in STA can be reduced to normal form in a number of β-reductions polynomial in its lenght. Moreover all polynomial time functions can be computed by terms typable in this system. So STA is correct and complete for (F)PTIME.

STA admits polynomial iterations, so we will use it as starting point for the characterization of polynomial space, through $\mathrm{STA}_{\mathbf{B}}$ [4], a type assignment system obtained from STA by increasing the set of types by a type $\mathbf{B}$ for booleans, and the λ-calculus by two boolean constants and a conditional constructor. $\mathrm{STA}_{\mathbf{B}}$ assigns types to terms in such a way that the evaluation of programs (closed terms of type $\mathbf{B}$) can be performed carefully in polynomial space. Moreover all polynomial space decision problems can be computed by programs of this system. So $\mathrm{STA}_{\mathbf{B}}$ is correct and complete for PSPACE.

The characterization of the complexity class NPTIME is made through the type assignment system STA_{+}. The λ-calculus is extended by a non-deterministic choice operator $+$, and STA_{+} is just STA plus by a rule for dealing with this new term constructor. The typing of the $+$ operator is inspired by the logical rule proposed by Maurel in order to deal with non determinism inside Light Affine Logic (LAL) [10]. Obviously the obtained language is no more confluent, but it preserves both

the properties of subject reduction and strong normalization, when the definition of strong normalization is revised taking into account that a term can have more than one normal form. If a term can be typed in STA_+, then each one of its normal forms can be reached in a number of reduction steps which is polynomial in the size of the term, if the reduction is performed carefully. Moreover we show that every non deterministic polynomial time decision problem can be simulated by a term typable in STA_+. Then STA_+ is correct and complete for NPTIME.
The soundness and completeness proofs for the three systems are based on the same methodology, in particular the completeness is proved, in all cases, by coding all Turing machines characterizing the different computational classes by terms typable in the corresponding type assignment system.
The paper is organized as follows. In Section 2 the Soft Linear Logic is recalled. In Section 3 the type assignments STA is presented, and the technical problems to be solved in order to derive it from SLL are briefly discussed. Section 4 contains the type assignment system $\mathrm{STA}_{\mathbf{B}}$. Moreover the two Sections 3 and 4 contain a sketch of the proof methodology used for proving the correctness and completeness of the two systems with respect to (F)PTIME and PSPACE respectively. In Section 5 the type assignment system STA_+ is presented, and its correctness and completeness with respect to NPTIME are proved.

2 Soft Linear Logic

Soft Linear Logic (SLL) has been introduced by Lafont [7], in order to capture the polynomial time complexity class PTIME. Here we will consider just the intuitionistic fragment with the connectives $\multimap, \forall$ and the modality !, being it sufficient for our aims.

Definition 2.1

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

$$U, V, Z ::= \alpha \mid U \multimap U \mid \forall \alpha.U \mid !U$$

where α ranges over a countable set of variables.

ii) A SLL context is a multiset of SLL formulae. Contexts are ranged over by Γ, Δ.

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

$$\Gamma \vdash U$$

where Γ is a context and U is a formula. The rules are given in Table 1.

SLL is a restriction of Girard's Linear Logic (LL) [3], obtained in two steps. First, by replacing the rules of LL dealing with the modality !:

$$\frac{!\Gamma \vdash U}{!\Gamma \vdash !U}\ (!R) \qquad \frac{\Gamma, V \vdash U}{\Gamma, !V \vdash U}\ (!L)$$

$$\frac{}{U \vdash U}\ (Id) \qquad \frac{\Gamma \vdash U \quad V, \Delta \vdash Z}{\Gamma, U \multimap V, \Delta \vdash: Z}\ (\multimap L) \qquad \frac{\Gamma \vdash U}{\Gamma \vdash \forall \alpha.U}\ (\forall R)$$

$$\frac{\Gamma \vdash U \quad \Delta, U \vdash V}{\Gamma, \Delta \vdash V}\ (cut) \qquad \frac{\Gamma, U \vdash V}{\Gamma \vdash U \multimap V}\ (\multimap R) \qquad \frac{\Gamma, U[V/\alpha] \vdash Z}{\Gamma, \forall \alpha.U \vdash Z}\ (\forall L)$$

$$\frac{\Gamma \vdash U}{!\Gamma \vdash !U}\ (sp) \qquad \frac{\Gamma, \overbrace{U, ..., U}^{n} \vdash V \quad n \geq 0}{\Gamma, !U \vdash V}\ (m)$$

Table 1
Soft Linear Logic

and the structural rules of weakening and contraction:

$$\frac{\Gamma \vdash U}{\Gamma, !V \vdash U}\ (W) \qquad \frac{\Gamma, !V, !V \vdash U}{\Gamma, !V \vdash U}\ (C)$$

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

$$\frac{\Gamma, \overbrace{U, \ldots, U}^{n \text{ times}} \vdash Z}{\Gamma, !U \vdash Z}\ (mpx) \qquad \frac{\Gamma \vdash U}{!\Gamma \vdash !U}\ (sp) \qquad \frac{\Gamma, !!V \vdash U}{\Gamma, !V \vdash U}\ (digging)$$

Note that the (mpx) 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 (mpx) followed by $(digging)$.
The second step is to erase the rule $(digging)$: the result is that there isn't anymore the linear correspondence

$$!U \multimap !!U.$$

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

In order to give the complexity properties of SLL, we need to recall some important notions. First of all, we assume that the reader knows the notion of proof-nets, a representation of proofs as graphs. Let π be a SLL proof-net: its *size* is the number of its nodes, its *rank* is the maximum rank of a multiplexor in it, and its *degree* is the maximum number of nested applications of rule (sp) in it. The cut elimination procedure applied on a proof-net π takes a number of steps $\leq |\pi| \times k^d$, where $|\pi|$ is the size of π, and k and d are its rank and degree, respectively. So, assuming the proof-nets as computational model, and the cut-elimination as computational rule, SLL is correct for polynomial time computations, once the degree is fixed. The

completeness is achieved through a coding of the polynomial time decision problems computed by a Turing Machines as SLL proof-nets and it is based on the fact that data (Church numerals and boolean strings, which are used for representing respectively iterators and input) can be coded as proof-nets with degree 0. So the following theorem holds.

Theorem 2.2 ([7]) SLL *is correct and complete for* PTIME.

3 A type assignment for (F)PTIME

A standard decoration of SLL proofs by λ-terms does not work for our purposes, since it has two main problems. First, subject reduction does not hold for it, since there is not a direct correspondence between cut-elimination in proofs and β-reduction. The problem is not typical of SLL, but it arises for all the decorations of light logics proofs by λ-terms made in the standard way. For a discussion about this problem, the reader can refer to [1], where it is discussed for the Light Affine Logic, and [5], where the problem for Soft Linear Logic is presented.
Very roughly speaking, SLL proof-nets have the property that only boxes can be duplicated. In the decorated calculus, a box corresponds to a λ-term typed by a modal type starting from all modal assumptions. So the property of substitution does not work in general, but it depends not only on types but also on the shape of the context. The unpleasant consequence is that the subject reduction does not hold. Moreover a type assignment for λ-calculus designed as decoration of a sequent calculus is not easy to deal with, since terms are built through substitutions, and so it is not possible to carry out proofs by induction on the structure of terms.
In this section we present STA, a type assignment system in natural deduction, based on SLL, which enjoys the subject reduction property, and which inherits from SLL the good computational properties. In order to obtain subject reduction, terms are built in a linear way, and possible duplications of subterms are obtained through the rules dealing with the modality, namely soft promotion and multiplexor. This is possible thanks to a restriction of the SLL formulae, given in the next definition.

Definition 3.1

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

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

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 all the free occurrences of $\mathtt{x}$ in $\mathtt{M}$ by $\mathtt{N}$. $\rightarrow_\beta^*$ is the reflexive and transitive closure of $\rightarrow_\beta$.

$$\frac{}{\mathtt{x} : A \vdash \mathtt{x} : A}\ (Ax) \qquad \frac{\Gamma \vdash \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : A \vdash \mathtt{M} : \sigma}\ (w)$$

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

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

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

Table 2
The Soft Type Assignment system STA

iii) The set $\mathbf{T}$ of *soft types* is defined as follows:

$$A, B, C ::= \alpha \mid \sigma \multimap A \mid \forall \alpha.A \quad \text{(Linear Types)}$$
$$\sigma, \tau, \rho, \mu, \nu ::= A \mid !\sigma$$

where α ranges over a countable set of type variables.

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

v) 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 a soft type. The rules are given in Table 2.

$\mathrm{FV}(\mathtt{M})$ denotes the set of free variables of the term $\mathtt{M}$, while $\mathrm{FTV}(\sigma)$ denotes the set of free variables of the type σ. Let Γ, Δ be contexts, then $\mathrm{dom}(\Gamma) = \{\mathtt{x} \mid \exists \mathtt{x} : \sigma \in \Gamma\}$, $\mathrm{FTV}(\Gamma) = \{\mathrm{FTV}(\sigma) \mid \exists \mathtt{x} : \sigma \in \Gamma\}$, $!\Gamma = \{\mathtt{x} :!\sigma \mid \exists \mathtt{x} : \sigma \in \Gamma\}$ and $\Gamma \# \Delta$ means $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$.
$\sigma[A/\alpha]$ denotes the capture free substitution of all occurrences of the type variable α by the linear type A: note that this kind of substitution preserves the correct syntax of types.
Proofs in STA are denoted by Π, Σ, Φ, Ψ. The statement $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$ denotes a proof Π with conclusion $\Gamma \vdash \mathtt{M} : \sigma$. As usual $\vdash \mathtt{M} : \sigma$ is a short for $\emptyset \vdash \mathtt{M} : \sigma$.
Some comments follow. STA is not a *true* natural deduction system, since rule (m) performs a substitution in the subject, but it replaces variables by variables, so the

size of the term is preserved, and induction on it can be used. Note that rule (Ax) and (w) introduce variables with linear type, rule $(\multimap I)$ introduces the binding λ on a term with a linear type and rule $(\multimap E)$ builds a term with a linear type. So, the only way of duplicating a subterm is by using the rule (sp).

Property 1 $\Pi \rhd \Gamma \vdash \mathtt{M} : !\sigma$ *implies* Π *can be transformed (commuting some rules) in a derivation of the shape:*

$$\frac{\Gamma' \vdash \mathtt{M} : \sigma}{!\Gamma' \vdash \mathtt{M} : !\sigma}\ (sp)$$

followed by a sequence of rules (w) *and* (m)*, working on variables not occurring in* $\mathtt{M}$*, that transforms the context* $!\Gamma'$ *in* Γ*.*

The consequence of the previous property is that a term with a modal type corresponds always to a box (but some rules with no computational meaning). So the following fundamental results hold:

Theorem 3.2

i) **(Subject reduction)** $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_\beta \mathtt{N}$ *imply* $\Gamma \vdash \mathtt{N} : \sigma$*.*

ii) **(Strong Normalization)** $\Gamma \vdash \mathtt{M} : \sigma$*, for some* Γ *and* σ*, implies* $\mathtt{M}$ *is strongly normalizing.*

As far as the complexity results are concerned, we will prove that STA is correct and complete for (F)PTIME. Namely every term typable in STA reduces to normal form in a number of β-reduction steps which is polynomial in the size of the term, and moreover every polynomial time Turing machine can be coded by a λ-term typable in STA.

3.1 Deterministic polynomial time soundness

In order to prove the complexity results, we need to define some measures both of the terms and of the type derivations. The formal definition of measures is in the Appendix. Roughly speaking, $|\mathtt{M}|$ and $|\Pi|$ are the *size* of the term $\mathtt{M}$ and of the derivation Π respectively, i.e., the number of symbols of $\mathtt{M}$ and the number of rules of Π. The *rank* of a multiplexor is the number of variables $\mathtt{x}_i$ in it, such that $\mathtt{x}_i \in \mathrm{FV}(\mathtt{M})$ (using the notation of Table 2). $\mathtt{rk}(\Pi)$, the *rank* of the derivation Π, is the maximum rank of a multiplexor rule in Π. The *weight* of Π with respect to r, $\mathtt{W}(\Pi, r)$, is a static upper bound of the size of all the proofs obtained during the normalization process of Π, in case every box is duplicated at most r times. Notice that, since the linear construction of terms by the system, the number of possible duplications is limited by the *degree* of Π, $\mathtt{d}(\Pi)$, which is the maximum nesting of applications of rule (sp) in it.
It is important to notice that the trasformation of proofs defined in the text of Property 1 never increases the measures of a derivation. The relations between these measures are shown in the next lemma.

Lemma 3.3 *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$*. Then:*

1. $\mathtt{rk}(\Pi) \leq |\mathtt{M}| \leq |\Pi|$.

2. $\mathtt{W}(\Pi, 1) \leq |\mathtt{M}|$.
3. $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)}\mathtt{W}(\Pi, 1)$
4. $\mathtt{x} :!^q A \in \Gamma$ *implies* $n_o(\mathtt{x}, \mathtt{M}) \leq (\mathtt{rk}(\Pi))^q$, *where* $n_o(\mathtt{x}, \mathtt{M})$ *denotes the number of free occurrences of* $\mathtt{x}$ *in* $\mathtt{M}$.

The weight of a proof decreases when a β-reduction is performed.

Lemma 3.4 *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_\beta \mathtt{M}'$. *There is a derivation* $\Pi' \rhd \Gamma \vdash \mathtt{M}' : \sigma$, *with* $\mathtt{rk}(\Pi) \geq \mathtt{rk}(\Pi')$, *such that if* $r \geq \mathtt{rk}(\Pi')$ *then* $\mathtt{W}(\Pi', r) < \mathtt{W}(\Pi, r)$.

As consequence, the following theorem holds.

Theorem 3.5 (Deterministic Polynomial Time Soundness) *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$, *then* $\mathtt{M}$ *can be evaluated to normal form in a number of* β*-reduction steps* $\in O(|\mathtt{M}|^{(\mathtt{d}(\Pi)+1)})$.

So every typing of $\mathtt{M}$ is an upper bound of its reduction complexity. Since every β-reduction step can be carried out on a Turing machine in a number of steps polynomial in the size of the term, we have the soundness with respect to PTIME.

3.2 Deterministic polynomial time completeness

The completeness is proved by showing that every deterministic Turing machine (DTM) working in polynomial time can be programmed by terms typable in STA.

Let $\circ$ denote composition. In particular $\mathtt{M} \circ \mathtt{N}$ stands for $\lambda\mathtt{z}.\mathtt{M}(\mathtt{Nz})$ and $\mathtt{M}_1 \circ \mathtt{M}_2 \circ \cdots \circ \mathtt{M}_n$ stands for $\lambda\mathtt{z}.\mathtt{M}_1(\mathtt{M}_2(\cdots(\mathtt{M}_n\mathtt{z})))$. Tensor product is definable as $\sigma \otimes \tau \doteq \forall\alpha.(\sigma \multimap \tau \multimap \alpha) \multimap \alpha$. In particular $\langle \mathtt{M}, \mathtt{N} \rangle$ stands for $\lambda\mathtt{x}.\mathtt{xMN}$. Note that projectors are definable as usual since STA is an affine system. n-ary tensor product can be easily defined through the binary one and we use σ^n to denote $\sigma \otimes \cdots \otimes \sigma$ n-times.

Natural numbers are represented by Church numerals, i.e. $\mathtt{n} \doteq \lambda\mathtt{s}.\lambda\mathtt{z}.\mathtt{s}^n(\mathtt{z})$. Terms defining successor, addition and multiplication are typable by indexed types $\mathbf{N}_i \doteq \forall\alpha.!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$, where $!^i\sigma$ is short for $\underbrace{!...!}_{i}\sigma$. We write $\mathbf{N}$ to mean $\mathbf{N}_1$. In particular the following holds for STA:

Lemma 3.6 *Let* P *be a polynomial and* $deg(P)$ *be its degree. Then there is a term* $\mathtt{P}$ *defining* P *typable as:*

$$\vdash \mathtt{P} :!^{deg(P)}\mathbf{N} \multimap \mathbf{N}_{2deg(P)+1}$$

As usual, booleans are represented by the terms: $\mathtt{0} \doteq \lambda\mathtt{xy}.\mathtt{x}$, $\mathtt{1} \doteq \lambda\mathtt{xy}.\mathtt{y}$ and $\mathtt{if\ x\ then\ M\ else\ N} \doteq \mathtt{xMN}$, where $\mathtt{0}$ and $\mathtt{1}$ denotes *true* and *false* respectively. The usual boolean functions are typable using the type $\mathbf{B} \doteq \forall\alpha.\alpha \multimap \alpha \multimap \alpha$.
Note that, for fresh $\mathtt{x}$, we can conclude $\Gamma, \Delta, \mathtt{x} : \mathbf{B} \vdash \mathtt{if\ x\ then\ M\ else\ N} : \sigma$ from $\Gamma \vdash \mathtt{M} : \sigma$ and $\Delta \vdash \mathtt{N} : \sigma$ only if $\Gamma \# \Delta$. (Here, the multiplicative context management

is crucial, and this differs from the systems Λ_+ and $\mathrm{STA}_{\mathbf{B}}$ that we shall study in short while.) In particular the following holds for STA:

Lemma 3.7 *Each boolean total function $f : \mathbb{B}^n \to \mathbb{B}^m$, where $n, m \geq 1$, can be λ-defined by a term* $\mathtt{f}$ *typable in* STA *as* $\vdash \mathtt{f} : \mathbf{B}^n \multimap \mathbf{B}^m$.

Strings of boolean are represented by terms of the shape $\lambda \mathtt{cz}.\mathtt{cb}_0(\cdots(\mathtt{cb}_n \mathtt{z})\cdots)$ where $\mathtt{b}_i \in \{\mathtt{0}, \mathtt{1}\}$. Such terms are typable by the indexed type $\mathbf{S}_i \doteq \forall\alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha$. Again, we write $\mathbf{S}$ to mean $\mathbf{S}_1$. Moreover there is a term $\mathtt{len}$ typable as $\vdash \mathtt{len} : \mathbf{S}_i \multimap \mathbf{N}_i$ that given a string of boolean returns its length. Note that the data types defined above can be typed in STA by derivations with degree 0. In what follows we will refer to the next definition.

Definition 3.8 A *decision problem* $\mathcal{D} : \{0,1\}^* \to \{0,1\}$ is definable in STA by a term $\mathtt{M}$ typable as $\vdash \mathtt{M} : !^n\mathbf{S}_m \multimap \mathbf{B}$ for some $m, n \in \mathbb{N}$ if and only if for each input string $s \in \{0,1\}^*$ definable by $\mathtt{s}$:

$$\mathcal{D}(s) = 0 \iff \mathtt{Ms} \to_\beta^* 0.$$

DTM configurations can be encoded by terms of the shape:

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

where $\mathtt{cb}_0^l \circ \ldots \circ \mathtt{cb}_n^l$ and $\mathtt{cb}_0^r \circ \ldots \circ \mathtt{cb}_n^r$ are respectively the left and right hand-side words of the DTM tape and $\mathtt{Q}$ is a tuple of length q encoding the state. Such terms can be typed using the indexed type:

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

We can define terms dealing with DTM configurations. In particular for every polynomial R we have a term: $\vdash \mathtt{Init}_{\mathtt{R}} :!^{deg(R)+1}\mathbf{S}_i \multimap \mathbf{TM}_{2deg(R)+1}$ that given an input string $\mathtt{s}$, returns a DTM configuration in the initial state q_0 with the tape of lenght $R(|\mathtt{s}|)$ filled by $\mathtt{s}$. Moreover we have a term: $\vdash \mathtt{Tr} : \mathbf{TM}_i \multimap \mathbf{TM}_i$ defining the transition function and a term: $\vdash \mathtt{Ext} : \mathbf{TM}_i \multimap \mathbf{B}$ that returns either $\mathtt{0}$ or $\mathtt{1}$ if the given configuration is accepting or rejecting respectively.
So we can build terms of the shape:

$$\mathtt{DTM}_{\mathtt{P}} \doteq \lambda \mathtt{s}.\mathtt{Ext}(\mathtt{P}\ (\mathtt{len}\ \mathtt{s})\ \mathtt{Tr}\ (\mathtt{Init}_{\mathtt{P}}\ \mathtt{s}))$$

defining problems decidable by DTM working in polynomial time P.

Theorem 3.9 (PTIME Completeness) *A decision problem $\mathcal{D}$ decidable by a* DTM $\mathcal{M}$ *in* polynomial time P *is λ-definable by a term* $\mathtt{DTM}_{\mathtt{P}}$ *typable in* STA *as:*

$$\vdash \mathtt{DTM}_{\mathtt{P}} :!^{deg(P)+2}\mathbf{S} \multimap \mathbf{B}$$

Analogously, we can define a term $\vdash \mathtt{Ext}_{\mathtt{F}} : \mathbf{TM}_i \multimap \mathbf{S}_i$ extracting a string from a configuration, so we can build terms of the shape:

$$\mathtt{DTM}^{\mathtt{F}}_{\mathtt{P}} \doteq \lambda \mathtt{s}.\mathtt{Ext}_{\mathtt{F}}(\mathtt{P}\ (\mathtt{len}\ \mathtt{s})\ \mathtt{Tr}\ (\mathtt{Init}_{\mathtt{P}}\ \mathtt{s}))$$

$$\frac{}{\vdash \mathtt{0} : \mathbf{B}}\ (\mathbf{B}_0 I) \qquad \frac{}{\vdash \mathtt{1} : \mathbf{B}}\ (\mathbf{B}_1 I)$$

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N_0} : A \quad \Gamma \vdash \mathtt{N_1} : A}{\Gamma \vdash \ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ : A}\ (\mathbf{B}E)$$

Table 3
The Soft Type Assignment system with Booleans

defining functions computable by DTM working in polynomial time P. So the completeness for FPTIME can also be achieved.

4 A type assignment for PSPACE

We present $\mathrm{STA}_{\mathbf{B}}$, a type assignment for polynomial space decision problem, which is built on the top of STA. For this, we add a type $\mathbf{B}$ for Booleans and we extend the language with a conditional construction $\mathtt{if\ M\ then\ N_0\ else\ N_1}$ with an additive typing rule.

Definition 4.1

(i) The set $\Lambda_{\mathcal{B}}$ of *terms* is defined by the following grammar:

$$\mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{V} ::= \mathtt{x} \mid \mathtt{0} \mid \mathtt{1} \mid \lambda \mathtt{x}.\mathtt{M} \mid \mathtt{MM} \mid \ \mathtt{if\ M\ then\ M\ else\ M}$$

where $\mathtt{x}$ ranges over a countable set of variables and $\mathtt{0}$ and $\mathtt{1}$ are *booleans*.

(ii) The set $\mathbf{T_B}$ of $\mathbf{B}$ *types* is defined as follows:

$$A ::= \mathbf{B} \mid \alpha \mid \sigma \multimap A \mid \forall \alpha.A \quad \text{(Linear Types)}$$

$$\sigma ::= A \mid !\sigma$$

where α ranges over a countable set of type variables and $\mathbf{B}$ is the only *ground* type.

The typing rules of $\mathrm{STA}_{\mathbf{B}}$ are the ones of STA with the typing rules for Boolean, which are written in Table 3.

The computational meaning of the λ-abstraction and of the conditional is conveyed by the reduction rule $\rightarrow_{\beta\delta}$.

Definition 4.2 The reduction relation $\rightarrow_{\beta\delta} \subseteq \Lambda_{\mathcal{B}} \times \Lambda_{\mathcal{B}}$ is the contextual closure of the following rules:

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

$$\mathtt{if\ 0\ then\ M\ else\ N} \ \rightarrow_{\delta} \mathtt{M}$$

$$\mathtt{if\ 1\ then\ M\ else\ N} \ \rightarrow_{\delta} \mathtt{N}$$

$\rightarrow^*_{\beta\delta}$ denotes the reflexive and transitive closure of $\rightarrow_{\beta\delta}$.

The other notions are very similar to the ones presented in the previous sections and we refer to them in this section.

Theorem 4.3

i) **(Subject reduction)** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$. *Then* $\Gamma \vdash \mathtt{N} : \sigma$.

ii) **(Strong Normalization)** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *then* M *is strongly normalizing with respect to the reduction relation* $\to_{\beta\delta}$.

4.1 Polynomial space soundness

Because of the additive rule ($\mathbf{B}E$), there is an exponential length derivation in $\mathrm{STA}_{\mathbf{B}}$. Indeed, consider the following term

$$(\lambda \mathtt{f}.\lambda \mathtt{z}.\mathtt{f}^n(\mathtt{z}))(\lambda \mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x})\mathtt{0}$$

A call by value strategy provides an exponential derivation length in n. However, a leftmost outermost computation with a careful bookkeeping evaluates the above term in polynomial time. This observation leads us to define an abstract machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$, which evaluates *programs*, i.e., closed terms of type **B**. The machine is reminiscent of Krivine's one [6], and is described in Table 4. There are two kinds of context. The first one is the m-context which is noted $\mathcal{A}$. It is used to store variable assignments which come from β-reductions, see the rules (β) of Table 4. So, a variable substitution is performed only if it is necessary, see the rule (h). The second kind of context is a **B**-context. The machine pushes nested conditionals in the **B**-context in order to be able to jump to the right conditional branch, avoiding thus useless and costly computation, see both rules (`if` 0) and (`if` 1). The two contexts are formally defined in the next definition.

Definition 4.4

(i) An *m-context* $\mathcal{A}$ is a sequence of variable assignments of the shape $\mathtt{x}_i := \mathtt{M}_i$ where all variables $\mathtt{x}_i$ are distinct ($1 \leq i \leq n$), and its *size*, $|\mathcal{A}|$, is $\sum_{1\leq i\leq n} |\mathtt{M}_i| + i$.

(ii) Let $\circ$ be a distinguished symbol. A **B**-*contexts* is defined by the following grammar:

$$\mathcal{C}[\circ] ::= \circ \mid (\ \mathtt{if}\ \mathcal{C}[\circ]\ \mathtt{then\ M\ else\ N}\ )\mathtt{V}_1 \cdots \mathtt{V}_n$$

and its size $|\mathcal{C}[\circ]|$ is the size of the term obtained by replacing the symbol $\circ$ by a variable.

A computation of the abstract machine is abbreviated by $\nabla :: \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ where ∇ is a derivation tree, in which each node is a configuration of the shape $\mathcal{C}', \mathcal{A}' \models \mathtt{N} \Downarrow \mathtt{b}'$. In particular, the conclusion of the derivation is the initial configuration. $\models \mathtt{M} \Downarrow b$ is a short for $[\circ], \varepsilon \models \mathtt{M} \Downarrow b$. The machine computes the $\to_{\beta\delta}$ reduction, as proved by the next lemma, where, if $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n : \mathtt{N}_n]$, $(\mathtt{M})^{\mathcal{A}}$ is $\mathtt{M}[\mathtt{N}_n/\mathtt{x}_n][\mathtt{N}_{n-1}/\mathtt{x}_{n-1}]\cdots[\mathtt{N}_1/\mathtt{x}_1]$. In particular the machine follows the leftmost outermost reduction strategy.

$$\frac{}{\mathcal{C}, \mathcal{A} \models \mathtt{b} \Downarrow \mathtt{b}}\ (Ax)$$

$$\frac{\mathcal{C}, \mathcal{A}@\{\mathtt{x}' := \mathtt{N}\} \models \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}^*}{\mathcal{C}, \mathcal{A} \models (\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\beta)$$

$$\frac{\{\mathtt{x} := \mathtt{N}\} \in \mathcal{A} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A} \models \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (h)$$

$$\frac{\mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\ \mathtt{if}\ \mathtt{0})$$

$$\frac{\mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{1} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\ \mathtt{if}\ \mathtt{1})$$

(*) $\mathtt{x}'$ is a fresh variable.

Table 4
The Abstract Machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$

Lemma 4.5 *i) Let* $\Pi \rhd \vdash \mathtt{M} : \mathbf{B}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$. *For each* $\mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$

$$\mathtt{M} \to^*_{\beta\delta} (\mathcal{C}[\mathtt{N}])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$$

ii) Let M *be a program, i.e.,* $\vdash \mathtt{M} : \mathbf{B}$. *Then* $\models \mathtt{M} \Downarrow \mathtt{b}$.

The size of a configuration $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ is the sum $|\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}|$. We define the space used by the abstract machine during the evaluation ∇ of a program to be the maximal size of a configuration in ∇. This space usage is clearly related to Turing machines. The next lemma gives a bound on the dimensions of all the components of a machine configuration, namely the term, the m-context and the **B**-context.

Lemma 4.6 *Let* M *be a closed term,* $\Pi \rhd \vdash \mathtt{M} : \mathbf{B}$ *where* $\mathtt{d}(\Pi) = d$, *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each intermediate machine configuration* $\mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$*:*

(i) $|\mathcal{A}| \leq 2|\mathtt{M}|^{d+2}$
(ii) $|\mathtt{N}| \leq 2|\mathtt{M}|^{2d+2}$
(iii) $|\mathcal{C}| \leq 2|\mathtt{M}|^{3d+3}$

The previous lemma is based on the observation that each term occurring in $\mathcal{A}$ is an instance of a subterm of M. Since, by Lemma 4.5, for each configuration $\mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$, $(\mathcal{C}[\mathtt{N}])^{\mathcal{A}}$ is a reduct of M and d is an upper bound on the number of subterms duplications during the reduction of M, the lemma follows. So we obtain

the following result.

Theorem 4.7 (Polynomial Space Soundness) *Let* $\Pi \triangleright \vdash \mathtt{M} : \mathbf{B}$. *Then* $\mathtt{M}$ *can be evaluated to normal form in space bounded by* $O(|\mathtt{M}|^{O(\mathtt{d}(\Pi))})$.

4.2 Polynomial space completeness

In what follows we use the fact that a polynomial space decision problem is computable by a polynomial time Alternating Turing Machine (ATM) and vice-versa [11,2]. We simulate a polynomial time ATM computation, following the line of [8] and [9], by a recursion with substitution of parameters encoded using higher type recursion. We consider the same representation of data types as in STA, in particular data types are typable through derivations with degree 0. It is worth noting that due to the presence of the $(\mathbf{B}E)$ rule it is possible to define the usual boolean connectives with an additive management of contexts.
An ATM configuration can be viewed as a DTM configuration with an extra information about the state. There are four kinds of state: *accepting* ($\mathtt{A}$), *rejecting* ($\mathtt{R}$), *universal* ($\wedge$), *existential* ($\vee$) . We can represent such information by tensor pairs of booleans. A configuration is accepting, rejecting, universal or existential depending on the kind of its state. We can encode ATM configurations by terms of the shape:

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

where, as in the case of a DTM, $\mathtt{cb}_0^l \circ \ldots \circ \mathtt{cb}_n^l$ and $\mathtt{cb}_0^r \circ \ldots \circ \mathtt{cb}_n^r$ are respectively the left and right hand-side words on the ATM tape, $\mathtt{Q}$ is a tuple of length q encoding the state and $\mathtt{k}$ is the tensor pair encoding the kind of state. Such terms can be typed using the following indexed type:

$$\mathbf{ATM}_i \doteq \forall \alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha)^2 \otimes \mathbf{B}^{q+2})$$

It is easy to adapt the terms dealing with DTM to the case of ATM. In particular:

$\vdash \mathtt{Init}_{\mathtt{R}} : !^{deg(R)+1}\mathbf{S}_i \multimap \mathbf{ATM}_{2deg(R)+1}$	Initial configuration
$\vdash \mathtt{Tr}_1, \mathtt{Tr}_2 : \mathbf{ATM}_i \multimap \mathbf{ATM}_i$	Transition functions
$\vdash \mathtt{Ext} : \mathbf{TM}_i \multimap \mathbf{B}$	Result State Extraction

Moreover we have a term: $\mathtt{Kind}$ typable as $\vdash \mathtt{Kind} : \mathbf{ATM_i} \multimap \mathbf{B}^2$ which takes a configuration and return its kind.
Given an ATM $\mathcal{M}$ working in polynomial time we define, a recursive evaluation procedure $\mathtt{eval}^{\mathtt{P}}_{\mathcal{M}}$, working in polynomial time P, which takes a string $\mathtt{s}$ and returns $\mathtt{0}$ or $\mathtt{1}$ if the initial configuration (with the tape filled with $\mathtt{s}$) leads to an accepting or rejecting configuration respectively. Without loss of generality we consider ATMs with transition relation of degree two (at each step we consider two transitions).
Using the conditional it is easy to define a function α such that: $\alpha(\mathtt{A}, \mathtt{M}_1, \mathtt{M}_2) = \mathtt{A}$, $\alpha(\mathtt{R}, \mathtt{M}_1, \mathtt{M}_2) = \mathtt{R}$, $\alpha(\wedge, \mathtt{M}_1, \mathtt{M}_2) = \mathtt{M}_1 \wedge \mathtt{M}_2$ and $\alpha(\vee, \mathtt{M}_1, \mathtt{M}_2) = \mathtt{M}_1 \vee \mathtt{M}_2$.
We would now define $\mathtt{eval}^{\mathtt{P}}_{\mathcal{M}}$ as an iteration of an higher order $\mathtt{Step}_{\mathcal{M}}$ function over

a Base case.

$$\mathtt{Base} \doteq \lambda \mathtt{c}.(\mathtt{Kind}\ \mathtt{c})$$

$$\mathtt{Step}_{\mathcal{M}} \doteq \lambda \mathtt{h}.\lambda \mathtt{c}.\alpha((\mathtt{Kind}\ \mathtt{c}), (\mathtt{h}(\mathtt{Tr}^1_{\mathcal{M}}\ \mathtt{c})), (\mathtt{h}(\mathtt{Tr}^2_{\mathcal{M}}\ \mathtt{c})))$$

It is easy to verify that such terms are typable as: $\vdash \mathtt{Base} : \mathbf{TM}_i \multimap \mathbf{B}^2$ and $\vdash \mathtt{Step}_{\mathcal{M}} : (\mathbf{TM}_i \multimap \mathbf{B}^2) \multimap \mathbf{TM}_i \multimap \mathbf{B}^2$ respectively. Hence, the evaluation function of an ATM $\mathcal{M}$ working in polynomial time P is definable as:

$$\mathtt{eval}^{\mathtt{P}}_{\mathcal{M}} \doteq \lambda \mathtt{s}.\mathtt{Ext}((\mathtt{P}\ (\mathtt{len}\ \mathtt{s})\ \mathtt{Step}_{\mathcal{M}}\ \mathtt{Base})(\mathtt{Init}_{\mathtt{P}}\ \mathtt{s}))$$

From the well known result of [2] we can conclude.

Theorem 4.8 (PSPACE Completeness) *A decision problem $\mathcal{D}$ decidable by a* DTM $\mathcal{M}$ *in* polynomial space P *is definable by a term* $\mathtt{eval}^{\mathtt{P}}_{\mathcal{M}}$ *typable in* $\mathrm{STA}_{\mathbf{B}}$ *as:*

$$\vdash \mathtt{eval}^{\mathtt{P}}_{\mathcal{M}} : !^{deg(P)+2}\mathbf{S} \multimap \mathbf{B}$$

The characterization of FPSPACE could be reached by enriching the calculus with strings of booleans and the type assignment system by a new type and rule for dealing with them. Moreover the machine could be extended to deal with closed terms typed by the type of the boolean strings. The extension does not present any particular problem, but it would be technically very boring.

5 A type assignment for NP

We here present STA_+, a type assignment system characterizing non deterministic polynomial time computations, which is built on the top of STA. We need to extend the language by a non deterministic construction $\mathtt{M} + \mathtt{N}$, and to deal carefully with the corresponding reduction rules, while the set of types remains unchanged.

Definition 5.1

i) The set Λ_+ of *terms* is defined by the following grammar:

$$\mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{V} ::= \mathtt{x} \mid \mathtt{MM} \mid \lambda \mathtt{x}.\mathtt{M} \mid \mathtt{M} + \mathtt{M}$$

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

ii) The Non Deterministic 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, and σ is a soft type. STA_+ is obtained by adding to the rules of STA the (*sum*) rule pictured in Table 5.

The following property still holds for STA_+.

$$\frac{\Gamma \vdash \mathtt{M} : A \quad \Gamma \vdash \mathtt{N} : A}{\Gamma \vdash \mathtt{M} + \mathtt{N} : A}\ (sum)$$

Table 5
The (*sum*) rule.

Property 2 $\Pi \rhd \Gamma \vdash \mathtt{M} : !\sigma$ *implies* Π *can be transformed (commuting some rules) in a derivation of the shape:*

$$\frac{\Gamma' \vdash \mathtt{M} : \sigma}{!\Gamma' \vdash \mathtt{M} : !\sigma}\ (sp)$$

followed by rules (w) *and* (m) *working on variables not occurring in* M.

The operational behaviour of Λ_+ is described in the following definition.

Definition 5.2 The reduction relation $\rightarrow_{\beta\gamma} \subseteq \Lambda_+ \times \Lambda_+$ is the contextual closure of the following rules:

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

$\rightarrow^*_{\beta\gamma}$ denotes the reflexive and transitive closure of $\rightarrow_{\beta\gamma}$.

The calculus Λ_+ equipped with the reduction $\beta\gamma$ is non confluent. In particular a term $\mathtt{M}$ can have more than one normal form. But we can extend to it in a natural way the notion of strong normalization. A term $\mathtt{M}$ is $\beta\gamma$-strongly normalizing if every reduction sequence starting from it stops. Then we can prove that the system enjoys the desired key properties.

Property 3

i) **(Subject reduction)** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \rightarrow_{\beta\gamma} \mathtt{N}$. *Then* $\Gamma \vdash \mathtt{N} : \sigma$.

ii) **(Strong Normalization)** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *then* $\mathtt{M}$ *is strongly normalizing with respect to the reduction relation* $\rightarrow_{\beta\gamma}$.

The subject reduction property can be immediately derived by the corresponding property of STA, since the definition of γ-reduction.

5.1 Non deterministic polynomial time soundness

We are now interested in proving the soundness of STA_+ with respect to non deterministic computations. In order to prove it, we need to proceed in a similar way as for PSPACE, and refer to a particular evaluation of terms. In fact, some $\beta\gamma$ reduction sequence can use both time and space exponential in the size of the initial term, as showed by the following example. Let $\mathtt{M} \equiv \underline{\mathtt{n}}((\lambda \mathtt{x}.\mathtt{zx} + \mathtt{zx})\mathtt{I})$, where $\underline{\mathtt{n}}$ is a Church numeral. If we first β-reduce $\mathtt{M}$ according to an innermost strategy, after $n+2$ steps we obtain a term with a number of γ-redexes which is exponential in n. In order to obtain the desired result, we need both to perform the reduction in an outermost

$$\frac{\mathcal{C}[\lambda \mathtt{x}.[\circ]], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{N}}{\mathcal{C}, \mathcal{A} \models \lambda \mathtt{x}.\mathtt{M} \Downarrow \lambda \mathtt{x}.\mathtt{N}}\ (\lambda) \qquad \frac{\mathcal{C}, \mathcal{A}@\{\mathtt{x}' := \mathtt{P}\} \models \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N} \qquad \mathtt{x}' \text{ fresh}}{\mathcal{C}, \mathcal{A} \models (\lambda \mathtt{x}.\mathtt{M})\mathtt{P}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}\ (\beta)$$

$$\frac{(*) \quad \mathcal{C}[\mathtt{x}[\circ]\mathtt{V}_2 \cdots \mathtt{V}_m], \mathcal{A} \models \mathtt{V}_1 \Downarrow \mathtt{N}_1 \quad \cdots \quad \mathcal{C}[\mathtt{x}\mathtt{N}_1 \cdots \mathtt{N}_{m-1}[\circ]], \mathcal{A} \models \mathtt{V}_m \Downarrow \mathtt{N}_m}{\mathcal{C}, \mathcal{A} \models \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{x}\mathtt{N}_1 \cdots \mathtt{N}_m}\ (h_1)$$

$$\frac{\{\mathtt{x} := \mathtt{P}\} \in \mathcal{A} \quad \mathcal{C}, \mathcal{A} \models \mathtt{P}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}{\mathcal{C}, \mathcal{A} \models \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}\ (h)$$

$$\frac{\mathcal{C}, \mathcal{A} \models \mathtt{M}_0\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}{\mathcal{C}, \mathcal{A} \models (\mathtt{M}_0 + \mathtt{M}_1)\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}\ (L) \qquad \frac{\mathcal{C}, \mathcal{A} \models \mathtt{M}_1\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}{\mathcal{C}, \mathcal{A} \models (\mathtt{M}_0 + \mathtt{M}_1)\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{N}}\ (R)$$

$$(*) \quad \mathtt{x} \notin \mathrm{dom}(\mathcal{A})$$

Table 6
The ND Abstract Machine $\mathrm{KND}^{\mathcal{C}}_{\mathcal{B}}$

way, and moreover to perform the substitutions, arised by the β-reduction, only when necessary. The abstract reduction machine $\mathrm{KND}^{\mathcal{C}}_{\mathcal{B}}$ defined in Table 6 does the required job. In fact every substitution is made, by the rule (h), only on the head occurrence of a variable, thanks to the m-context $\mathcal{A}$. Moreover, thanks to the rules (L) and (R) a γ-reduction cannot be postponed, but it is immediately performed when it appears in head position. As far as the space of an evaluation is concerned, this can be measured thanks to the context $\mathcal{C}$, which is defined by the following grammar:

$$\mathcal{C} ::= [\circ] \mid \lambda \mathtt{x}.\mathcal{C}[\circ] \mid \mathtt{x}\mathtt{M}_1 ... \mathtt{M}_i[\circ]\mathtt{M}_{i+1} ... \mathtt{M}_n \qquad (1 \leq i \leq n)$$

Note that, while in the PSPACE case we used a machine reducing only well typed terms, this new machine works on all Λ_+, and so it can run forever, in some cases. The strong normalization of terms in STA_+ assures us that for a well typed term the machine always stops. An example of a running of the machine is given in Table 7. The following lemma is the key tool for the soundness proof. We will freely use the notions and notations introduced in the previous section.

Lemma 5.3

i) $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{N}$ *and* $\mathcal{C}, \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{N}' \in \nabla$ *imply* $\mathtt{M} \to^*_{\beta\gamma} (\mathcal{C}[\mathtt{M}'])^{\mathcal{A}} \to^*_{\beta\gamma} (\mathcal{C}[\mathtt{N}'])^{\mathcal{A}} \to^*_{\beta\gamma} \mathtt{N}$ *where* $\mathtt{N}$ *is a normal form of* $\mathtt{M}$.

ii) $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{N}$ *and* $\mathcal{C}, \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{N}' \in \nabla$ *imply* $(\mathtt{M}')^{\mathcal{A}} \to^*_{\beta\gamma} \mathtt{N}'$ *where* $\mathtt{N}'$ *is a normal form.*

Proof. Easy, by inspecting the rules of the machine. □

The above lemma can be further refined.

Lemma 5.4 $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{N}$ *implies* $\mathtt{M} \to^*_{\beta\gamma} \mathtt{N}$ *where* $\mathtt{N}$ *is a normal form of* $\mathtt{M}$. *The computation can be carried out in a number of* $\beta\gamma$*-reduction equal to number of applications of rules* $(\beta), (L)$ *and* (R) *in* ∇.

Proof. Easy, by inspecting the rules of the machine. □

The analogous of Lemma 3.3 holds, thanks to a careful definition of the weight.

Lemma 5.5 *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$. *Then:*

1. $\mathtt{rk}(\Pi) \leq |\mathtt{M}| \leq |\Pi|$.
2. $\mathtt{W}(\Pi, 1) \leq |\mathtt{M}|$.
3. $\mathtt{W}(\Pi, r) \leq r^{\mathtt{d}(\Pi)}\mathtt{W}(\Pi, 1)$.
4. $\mathtt{M} \in \Lambda$ *implies that for every* $r \geq 1$*:* $|\mathtt{M}| \leq \mathtt{W}(\Pi, r)$.

The following lemma relates the measures on type derivation and the dimension of the computation in the machine.

Lemma 5.6 $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$ *and* $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{N}$ *imply that the number of rules applications in* ∇ *(but the rule* (h)*) is bounded by* $\mathtt{W}(\Pi, r)$ *for every* $r \geq rk(\Pi)$.

Proof. In every rule but (h) the weight of the conclusion is strictly grater than the weight of the premises. □

The above lemma leads to the following.

Lemma 5.7 *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$. *Then* $\mathtt{M}$ *can be evaluated to* every one *of its normal form in a number of* $\beta\gamma$*-reduction steps* $\in O(|\mathtt{M}|^{(\mathtt{d}(\Pi)+1)})$.

In order to extend the above lemma to a polytime result we need to consider also the space used in the computation. This is one of the reasons why we have introduced the machine $\mathrm{KND}^{\mathcal{C}}_{\mathcal{B}}$.
The non deterministic behaviour of $+$, together with the outermost reduction, imply that some occurrences of variables will be never replaced during the evaluation. E.g., in the term $(\lambda\mathtt{x}.\mathtt{x} + \mathtt{x})\mathtt{M}$ only one occurrence of $\mathtt{x}$ will be replaced by $\mathtt{M}$. The notion of *effective occurrence* gives an upper bound on the number of variables which could be replaced.

Definition 5.8 The number of *effective occurrences* $n_{eo}(\mathtt{x}, \mathtt{M})$ of the variable $\mathtt{x}$ occurring free in $\mathtt{M}$ is defined as:

$$n_{eo}(\mathtt{x}, \mathtt{x}) = 1, \quad n_{eo}(\mathtt{x}, \mathtt{y}) = 0, \quad n_{eo}(\mathtt{x}, \mathtt{M} + \mathtt{N}) = \max\{n_{eo}(\mathtt{x}, \mathtt{M}), n_{eo}(\mathtt{x}, \mathtt{N})\}$$

$$n_{eo}(\mathtt{x}, \mathtt{MN}) = n_{eo}(\mathtt{x}, \mathtt{M}) + n_{eo}(\mathtt{x}, \mathtt{N}), \quad n_{eo}(\mathtt{x}, \lambda\mathtt{y}.\mathtt{M}) = n_{eo}(\mathtt{x}, \mathtt{M}),$$

A type derivation gives us some informations about the number of effective occurrences of a free variable $\mathtt{x}$ in its subject $\mathtt{M}$.

Lemma 5.9 *Let* $\Pi \rhd \Gamma, \mathtt{x} :!^n A \rhd \mathtt{M} : \sigma$ *then* $n_{eo}(\mathtt{x}, \mathtt{M}) \leq \mathtt{rk}(\Pi)^n$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{C}_1, \mathcal{A}_1 \models \mathtt{y} \Downarrow \mathtt{y}}}{\mathcal{C}_0, \mathcal{A}_1 \models \lambda\mathtt{y}.\mathtt{y} \Downarrow \lambda\mathtt{y}.\mathtt{y}}}{\mathcal{C}_0, \mathcal{A}_1 \models \mathtt{x}_2 \Downarrow \mathtt{I}}}{\mathcal{C}_0, \mathcal{A}_0 \models \mathtt{II} \Downarrow \mathtt{I}}}{[\circ], \mathcal{A}_0 \models \mathtt{w}(\mathtt{II}) \Downarrow \mathtt{wI}}}{[\circ], \mathcal{A}_0 \models \mathtt{x}_1 \Downarrow \mathtt{wI}}}{[\circ], \mathcal{A}_0 \models \mathtt{x}_1 + \mathtt{x}_1 \Downarrow \mathtt{wI}}}{[\circ], \epsilon \models (\lambda\mathtt{x}.\mathtt{x} + \mathtt{x})(\mathtt{w}(\mathtt{II})) \Downarrow \mathtt{wI}}$$

$$\mathtt{I} = \lambda\mathtt{x}.\mathtt{x}$$
$$\mathcal{A}_0 = [\mathtt{x}_1 := \mathtt{w}(\mathtt{II})]$$
$$\mathcal{A}_1 = \mathcal{A}_0@[\mathtt{x}_2 := \lambda\mathtt{y}.\mathtt{y}]$$
$$\mathcal{C}_0 = \mathtt{w}[\circ]$$
$$\mathcal{C}_1 = \mathtt{w}(\lambda\mathtt{y}.[\circ])$$

Table 7
An example of computation in $\mathrm{KND}^{\mathcal{C}}_{\mathcal{B}}$.

The above lemma, by giving a bound on the number of effective occurrence is useful to give a bound on the number of applications of the (h) rule in every computation.
The size of a configuration $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{N}$ is the sum $|\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}| + |\mathtt{N}|$. We define the space of the abstract machine to be the maximal size of a configuration in a computation.

Lemma 5.10 *Let* $\mathtt{M}$ *be a closed term,* $\Pi \rhd \vdash \mathtt{M} : \mathbf{B}$ *where* $\mathtt{d}(\Pi) = d$*, and* $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{N}$*. Then for each intermediate machine configuration* $\mathcal{C}, \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{N}'$*:*

(i) $|\mathcal{A}| \leq 2|\mathtt{M}|^{d+2}$
(ii) $|\mathtt{M}'| \leq 2|\mathtt{M}|^{2d+2}$
(iii) $|\mathtt{N}'| \leq |\mathtt{M}|^{d+1}$
(iv) $|\mathcal{C}| \leq 2|\mathtt{M}|^{4d+4}$

Proof. Note that analogously to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ of the previous section also for $\mathrm{KND}^{\mathcal{C}}_{\mathcal{B}}$ each term occurring in $\mathcal{A}$ is an instance of a subterm of $\mathtt{M}$.

(i) It follows directly by the above observation and by Lemma 5.6.
(ii) It follows from the previous point and the fact that Lemma 5.9 gives a bound on the number of applications of the (h) rule on every variable in $\mathcal{A}$.
(iii) It follows from the fact that $\mathtt{N}'$ is a normal form so it is a term in Λ and we can apply Lemma 5.5.4.
(iv) It follows from the fact that the only rules that make the context $\mathcal{C}$ grow are the rules (λ) and (h_1). The case of (λ) rule is simple while the previous point can be used in the case of rule (h_1).

□

As consequence, the following lemma holds.

Lemma 5.11 *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$*, Then* $\mathtt{M}$ *can be evaluated to normal form in space*

bounded by $O(|\mathtt{M}|^{O(\mathtt{d}(\Pi))})$.

From the above lemma and Lemma 5.7 we can conclude the following.

Theorem 5.12 (Non Deterministic Polynomial Time Soundness) *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$, *then* $\mathtt{M}$ *can be evaluated to* every one *of its normal forms by a non deterministic Turing machine in time* $\in O(|\mathtt{M}|^{O(\mathtt{d}(\Pi))})$.

5.2 Non deterministic polynomial time completeness

We consider the same representation of data types as in STA, in particular data types are typable through derivations with degree 0. The following is the adaptation to the case of STA_+ of Definition 3.8.

Definition 5.13 A *decision problem* $\mathcal{D} : \{0,1\}^* \to \{0,1\}$ is definable in STA_+ by a term $\mathtt{M}$ typable as $\vdash \mathtt{M} :!^n\mathbf{S}_m \multimap \mathbf{B}$ for some $m, n \in \mathbb{N}$ if and only if for each input string $s \in \{0,1\}^*$ definable by $\mathtt{s}$:

$$\mathcal{D}(s) = 0 \iff \text{there exists a } \beta\gamma\text{-normal form of } \mathtt{M} \text{ equal to } \mathtt{0}.$$

A configuration of a non-deterministic Turing machine (NDTM) is identical to a configuration of a deterministic Turing machine. So we can encode NDTM configurations by terms of the shape:

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

which can be typed again using the indexed type:

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

Analogously to the case of a DTM, we have terms for dealing with NDTM configurations. In particular we have the following terms introduced in Section 3.1.

$\vdash \mathtt{Init}_\mathtt{R} :!^{deg(R)+1}\mathbf{S}_i \multimap \mathbf{TM}_{2deg(R)+1}$	Initial configuration
$\vdash \mathtt{Ext} : \mathbf{TM}_i \multimap \mathbf{B}$	Result State Extraction
$\vdash \mathtt{Ext}_\mathtt{F} : \mathbf{TM}_i \multimap \mathbf{S}_i$	Result Tape Extraction

In fact, what distinguish a NDTM by a DTM is that the behaviour of a NDTM is determined by a transition *relation* while the behaviour of a DTM is determined by a transition *function*. We have seen that every transition function is definable by a term typable as $\vdash \mathtt{Tr} : \mathbf{TM}_i \multimap \mathbf{TM}_i$. Hence, we can define a transition relation by a term $\mathtt{NDTr}$ defined as:

$$\mathtt{NDTr} \doteq \mathtt{Tr}_1 + \cdots + \mathtt{Tr}_n$$

Obviously the case $n = 1$ coincide with the case of a transition function. The term $\mathtt{NDTr}$ is typable using the (*sum*) rule as:

$$\vdash \mathtt{NDTr} : \mathbf{TM}_i \multimap \mathbf{TM}_i$$

So we can build terms of the shape:

$$\mathtt{NDTM_P} \doteq \lambda \mathtt{s.Ext(P\ (len\ s)\ NDTr\ (Init_P\ s))}$$

defining problems decidable by NDTM working in polynomial time P.

Theorem 5.14 (NPTIME Completeness) *A decision problem $\mathcal{D}$ decidable by a* NDTM $\mathcal{M}$ *in* polynomial time P *is definable by a term* $\mathtt{NDTM_P}$ *typable in* STA_+ *as:*

$$\vdash \mathtt{NDTM_P} : !^{deg(P)+2}\mathbf{S} \multimap \mathbf{B}$$

References

[1] P. Baillot and K. 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.

[2] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

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

[4] M. Gaboardi, J.-Y. Marion, and S. 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*, 2008. to appear.

[5] M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for λ-calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *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.

[6] J.-L. Krivine. A call-by-name lambda calculus machine. *Higher Order and Symbolic Computation*, 2007. To appear.

[7] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.

[8] D. Leivant and J-Y. Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. In *Proceedings of the 8th International Workshop on Computer Science Logic* (CSL '94), volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994.

[9] D. Leivant and J.-Y. Marion. Predicative functional recurrence and poly-space. In *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 369–380. Springer-Verlag, 1997.

[10] F. Maurel. Nondeterministic light logics and NP-time. In Martin Hofmann, editor, *Proceedings of the 6th International Conference on Typed Lambda-Calculi and Applications* (TLCA '03), volume 2701 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2003.

[11] W. J. Savitch. Relationship between nondeterministic and deterministic tape classes. *Journal of Computer and System Sciences*, 4:177–192, 1970.

A Measures definition

Definition A.1

- The *size* $|\mathtt{M}|$ of a term $\mathtt{M}$ is defined as $|\mathtt{x}| = 1$, $|\lambda \mathtt{x.M}| = |\mathtt{M}| + 1$, $|\mathtt{MN}| = |\mathtt{M} + \mathtt{N}| = |\mathtt{M}| + |\mathtt{N}| + 1$, $|\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ | = |\mathtt{M}| + |\mathtt{N_0}| + |\mathtt{N_1}| + 1$.
- The *size* $|\Pi|$ of a proof Π is the number of rules in Π.

- The *rank* of a rule (m), as defined in Table 2,

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

  is the number $k \leq n$ of variables $\mathtt{x}_i$ such that $\mathtt{x}_i \in \mathrm{FV}(\mathtt{M})$ $(1 \leq i \leq n\ )$. Let r be the the maximum rank of a rule (m) in Π. The *rank* $\mathtt{rk}(\Pi)$ of Π is the maximum between 1 and r.
- The *degree* $\mathtt{d}(\Pi)$ of Π is the maximum nesting of applications of rule (sp) in 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 $(Ax), (\mathbf{B}_0 I), (\mathbf{B}_1 I)$ then $\mathtt{W}(\Pi, r) = 1$.
  · If the last applied rule is:

$$\frac{\Sigma \rhd \Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : A}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A} \ (\multimap I)$$

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

$$\frac{\Sigma \rhd \Gamma \vdash \mathtt{M} : \sigma}{!\Gamma \vdash \mathtt{M} :! \sigma} \ (sp)$$

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

$$\frac{\Sigma \rhd \Gamma \vdash \mathtt{M} : \mu \multimap A \quad \Theta \rhd \Delta \vdash \mathtt{N} : \mu}{\Gamma, \Delta \vdash \mathtt{MN} : A} \ (\multimap E)$$

  then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r) + 1$.
  · If the last applied rule is:

$$\frac{\Sigma \rhd \Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Theta_0 \rhd \Gamma \vdash \mathtt{N}_0 : A \quad \Theta_1 \rhd \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ : A}$$

  then $\mathtt{W}(\Pi, r) = \max\{\mathtt{W}(\Sigma, r), \mathtt{W}(\Theta_0, r), \mathtt{W}(\Theta_1, r)\} + 1$
  · If the last applied rule is:

$$\frac{\Sigma \rhd \Gamma \vdash \mathtt{N}_0 : A \quad \Theta \rhd \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \mathtt{N}_0 + \mathtt{N}_1 : A}$$

  then $\mathtt{W}(\Pi, r) = \max\{\mathtt{W}(\Sigma, r), \mathtt{W}(\Theta_1, r)\} + 1$
  · In every other case $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r)$ where Σ is the unique premise derivation.