An Implicit Characterization of PSPACE

MARCO GABOARDI, Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna - INRIA Focus team - Mura Anteo Zamboni 7, 40127 Bologna, Italy, gaboardi@cs.unibo.it
JEAN-YVES MARION, Nancy-University, ENSMN-INPL, Loria - B.P. 239, 54506 Vandoeuvre-lès-Nancy, France, jean-yves.marion@loria.fr
SIMONA RONCHI DELLA ROCCA, Dipartimento di Informatica, Università degli Studi di Torino - Corso Svizzera 185, 10149 Torino, Italy, ronchi@di.unito.it

We present a type system for an extension of lambda calculus with a conditional construction, named $\mathrm{STA}_{\mathbf{B}}$, that characterizes the PSPACE class. This system is obtained by extending STA, a type assignment for lambda-calculus inspired by Lafont's Soft Linear Logic and characterizing the PTIME class. We extend STA by means of a ground type and terms for booleans and conditional. The key issue in the design of the type system is to manage the contexts in the rule for conditional in an additive way. Thanks to this rule, we are able to program polynomial time Alternating Turing Machines. From the well-known result APTIME = PSPACE, it follows that $\mathrm{STA}_{\mathbf{B}}$ is complete for PSPACE.
Conversely, inspired by the simulation of Alternating Turing machines by means of Deterministic Turing machine, we introduce a call-by-name evaluation machine with two memory devices in order to evaluate programs in polynomial space. As far as we know, this is the first characterization of PSPACE that is based on lambda calculus and light logics.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Machine-independent Complexity*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Type Structure*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda Calculus and Related Systems*; *Proof Theory*

General Terms: Languages, Theory, Design

Additional Key Words and Phrases: Implicit Computational Complexity, Linear Logic, Operational Semantics, Polynomial Space, Type Assignment

1. INTRODUCTION

The argument of this paper fits in the so called Implicit Computational Complexity area, whose aim is to provide complexity control through language restrictions, without using explicit machine models or external measures. In this setting, we are interested in the design of programming languages with bounded computational complexity. We want to use a ML-like approach, so having a λ-calculus like language, and a type assignment system for it, where the types guarantee, besides the functional correctness, also complexity properties. So, types can be used in a static way in order to check the correct behaviour of the programs, also with respect to the resource usage. According to these lines, we design in this paper a language correct and complete with respect to PSPACE. Namely, we supply, besides the calculus, a type assignment system and an evaluation machine, and we prove that well typed programs can be evaluated

Work partially supported by the projects ANR-08-BLANC-0211-01 "COMPLICE", MIUR-PRIN'07 "CONCERTO" and INRIA Associated Team CRISTAL.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YYYY ACM 1529-3785/YYYY/01-ARTA $10.00
DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

by the machine in polynomial space, and moreover that all decision functions computable in polynomial space can be coded by well typed programs.

Light Logics and type systems Several type systems characterizing complexity classes have been proposed so far [Leivant and Marion 1993; Bellantoni et al. 2000; Hofmann 1999; Baillot and Terui 2004; Danner and Royer 2006; Gaboardi and Ronchi Della Rocca 2007; Dal Lago and Schöpp 2010; Baillot et al. 2010].

We are here mainly concerned with using ideas from Linear Logic to obtain type assignment systems for λ-calculus. In this setting, some proposal have been made but they are quite all related to time complexity. The key idea is to use as types the formulae of the light logics, which characterize some classes of time complexity: Light Linear Logic (LLL) of Girard [Girard 1998], and Soft Linear Logic (SLL) of Lafont [Lafont 2004] characterize polynomial time, while Elementary Linear Logic (ELL) [Girard 1998] characterizes elementary time. The characterization is based on the fact that cut-elimination on proofs in these logics is performed in a number of steps which depends in a polynomial or elementary way from the initial size of the proof (while the degree of the proof, i.e., the nesting of exponential rules, is fixed). Moreover, the size of each proof in the cut elimination process can be bound by a polynomial or an elementary function in the initial size of the proof, respectively. In addition, all these logics are also complete with respect to the related complexity class, using proof-nets for coding functions.

The good properties of such logics have been fruitfully used in order to design type assignment systems for λ-calculus which are correct and complete with respect to the polynomial or elementary time complexity bound. Namely, every well typed term β-reduces to normal form in a number of steps that depends in a polynomial or elementary way from its size, and moreover all functions with the corresponding complexity are representable by a well typed term. Examples of polynomial time type assignment systems are in [Baillot and Terui 2004; 2009] and [Gaboardi and Ronchi Della Rocca 2007; 2009], based respectively on LAL (an affine variant of LLL designed by Asperti and Roversi [Asperti and Roversi 2002]) and on SLL. Moreover, an example of an elementary type assignment system is in [Coppola et al. 2005; 2008].

Contribution Our starting point is the fact that polynomial space computations coincide with polynomial time Alternating Turing Machine computations (APTIME). In particular, by the results in [Savitch 1970] and [Chandra et al. 1981], it follows

$$\text{PSPACE} = \text{NPSPACE} = \text{APTIME}$$

So, we start from the type assignment system STA for λ-calculus introduced in [Gaboardi and Ronchi Della Rocca 2007]. It is based on SLL, in the sense that in STA both types are a proper subset of SLL formulae, and type assignment derivations correspond, through the Curry-Howard isomorphism, to a proper subset of SLL derivations. STA is correct and complete (in the sense said before) with respect to polynomial time computations.

Then, we design the language $\Lambda_{\mathcal{B}}$, which is an extension of λ-calculus with two boolean constants and a conditional constructor, and we supply it by a type assignment system ($\text{STA}_{\mathbf{B}}$), where the types are STA types plus a constant type $\mathbf{B}$ for booleans, and rules for conditional. In particular, the elimination rule for conditional is the following:

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N}_0 : A \quad \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A} \ (\mathbf{B}E)$$

In this rule, contexts are managed in an additive way, that is with free contractions. From a computational point of view, this intuitively means that a computation can repeatedly fork into subcomputations and the result is obtained by a backward computation from all subcomputation results.

While the time complexity result for STA is not related to a particular evaluation strategy, here, for characterizing space complexity, the evaluation should be done carefully. Indeed, an uncontrolled evaluation can construct exponential size terms. So we define a call-by-name evaluation machine, $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$, inspired by Krivine's machine [Krivine 2007] for λ-calculus, where substitutions are made only on head variables. This machine is equipped with two memory devices, and the space used by it is proved to be the dimension of its maximal configuration. The proof is made through the design of an equivalent small step machine. Then, we prove that, if $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ takes a program (i.e., a closed term well typed by a ground type) as input, then the size of each configuration is polynomially bounded in the size of the input. From this it follows that every program can be evaluated by a Turing Machine in polynomial space. Conversely, we encode every polynomial time Alternating Turing Machine by a program well typed in $\mathrm{STA}_{\mathbf{B}}$. The simulation relies on a higher order representation of a parameter substitution recurrence schema inspired by the one in [Leivant and Marion 1994].

Related works The present work extends the preliminary results that have been presented to POPL '08 [Gaboardi et al. 2008a]. The system $\mathrm{STA}_{\mathbf{B}}$ is the first characterization of PSPACE through a type assignment system in the light logics setting. A proposal for a similar characterization has been made by Terui [Terui 2000], but the work has never been completed.

The characterization presented here is strongly based on the additive rule $(\mathbf{B}E)$ presented above. The key role played by this rule in the characterization of the PSPACE class has been independently suggested by Hofmann in the context of non-size-increasing computations [Hofmann 2003]. There, the author showed that by adding to his LFPL language a form of *restricted duplication* one can encode the "quantified boolean formulas problem" and recover exactly the behaviour of the rule $(\mathbf{B}E)$. Besides the difference in the setting where our study is developed with respect to the Hofmann's one, our work differs from this also in the fact that we give a concrete syntactic proof of PSPACE soundness for programs by means of an evaluation machine that is close to a possible implementation. Hofmann's PSPACE soundness instead relies on a semantic argument that by being abstract hides the technical difficulties that one needs to deal with in the evaluation of programs. Moreover, instead of encoding quantified boolean formulas as done by Hofmann, we here give a PSPACE completeness result based on the definability of all polynomial time Alternating Turing Machines.

In our characterization we make use of boolean constants in order to have a fine control of the space needed to evaluate programs. A use of constants similar in spirit to the present one has been also employed by the second author in [Leivant and Marion 1993], in order to give a characterization of the PTIME class.

There are several other implicit characterizations of polynomial space computations using principles that differ from the ones explored in this paper. The characterizations in [Leivant and Marion 1994; 1997] and [Oitavem 2001; 2008] are based on ramified recursions over binary words. Schubert has shown that the β-reduction of simply typed λ-terms containing redexes of order at most three is complete for PSPACE [Schubert 2001]. In finite model theory, PSPACE is captured by first order queries with a partial fixed point operator [Vardi 1982; Abiteboul and Vianu 1989]. The reader may consult the recent book [Grädel et al. 2007]. Finally there are some algebraic characterizations like the one [Goerdt 1992] or [Jones 2001] but which are, in essence, over finite domains.

Apart from the class PSPACE, the light logic principles have been used to characterize other interesting complexity classes. As already stressed, the light logics have been used to characterize the class PTIME [Girard 1998; Lafont 2004] and the class of Elementary functions [Girard 1998]. Moreover, in [Maurel 2003] and [Gaboardi et al. 2008b] an explicit sum rule to deal with non deterministic computation has been studied in the setting of Light Linear Logic and Soft Linear Logic, respectively. Both these works give implicit characterizations of the class NPTIME. Another important work in this direction is the one in [Schöpp 2007] where a logical system characterizing logarithmic space computations is defined, the Stratified Bounded Affine Logic (SBAL). Interestingly, the logarithmic space soundness for SBAL is proved in an interactive way by means of a geometry of interaction algorithm considering only proofs of certain sequents to represent the functions computable in logarithmic space. This idea was already present in the previous work [Schöpp 2006] of the same author and it has been further explored in the recent work [Dal Lago and Schöpp 2010].

Outline of the paper In Section 2 the system $\mathrm{STA}_{\mathbf{B}}$ is introduced and the proofs of the subject reduction and of the strong normalization properties are given. The strong normalization is proved by using a translation from $\mathrm{STA}_{\mathbf{B}}$ to System F terms. In Section 3 the operational semantics of $\mathrm{STA}_{\mathbf{B}}$ programs is defined. Two abstract machines $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ and $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$ are given. One implements a big step evaluation while the other implements a small step reduction. The computations in the two machines are then proved to be equivalent. In Section 4 we show that $\mathrm{STA}_{\mathbf{B}}$ programs can be executed in polynomial space. This is shown by establishing some formal relations between the configurations in the big step machine computations and the type derivations in $\mathrm{STA}_{\mathbf{B}}$. In this way, the bound on the size of each computation can be extracted from the information inherited from the type derivation. In Section 5 the completeness for PSPACE is proved by showing that the Alternating Turing Machines working in polynomial time can be programmed using $\mathrm{STA}_{\mathbf{B}}$ programs. Finally, Section 6 contains some conclusions.

2. SOFT TYPE ASSIGNMENT SYSTEM WITH BOOLEANS

In this section we present the paradigmatic language $\Lambda_{\mathcal{B}}$ and a type assignment for it, $\mathrm{STA}_{\mathbf{B}}$, and we will prove that $\mathrm{STA}_{\mathbf{B}}$ enjoys the properties of subject reduction and strong normalization. $\Lambda_{\mathcal{B}}$ is an extension of the λ-calculus with boolean constants $\mathtt{0}, \mathtt{1}$ and an `if` constructor. $\mathrm{STA}_{\mathbf{B}}$ is an extension of the type system STA for λ-calculus introduced in [Gaboardi and Ronchi Della Rocca 2007], which assigns to λ-terms a proper subset of formulae of Lafont's Soft Linear Logic [Lafont 2004], and it is correct and complete for polynomial time computations.

Definition 2.1 ($\Lambda_{\mathcal{B}}$).

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

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

where $\mathtt{x}$ ranges over a countable set of variables and $\mathcal{B} = \{\mathtt{0}, \mathtt{1}\}$ is the set of *booleans*.

(2) The *reduction relation* $\rightarrow_{\beta\delta} \subseteq \Lambda_{\mathcal{B}} \times \Lambda_{\mathcal{B}}$ is the contextual closure of the following rules:

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

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

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

$\rightarrow^{+}_{\beta\delta}$ denotes the transitive closure of $\rightarrow_{\beta\delta}$, the notation $\rightarrow^{*}_{\beta\delta}$ stands for the transitive and reflexive closure of $\rightarrow_{\beta\delta}$, while $=_{\beta\delta}$ denotes its transitive, reflexive and symmetric closure.

(3) The *size* of a term $\mathtt{M}$ is denoted as $|\mathtt{M}|$ and is defined inductively as

$$|\mathtt{x}| = |\mathtt{0}| = |\mathtt{1}| = 1 \qquad |\lambda\mathtt{x}.\mathtt{M}| = |\mathtt{M}|+1 \qquad |\mathtt{MN}| = |\mathtt{M}| + |\mathtt{N}|$$

$$|\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ | = |\mathtt{M}| + |\mathtt{N}_0| + |\mathtt{N}_1| + 1$$

Note that we use the term $\mathtt{0}$ to denote "true" and the term $\mathtt{1}$ to denote "false".

NOTATION 1. *Terms are denoted by* $\mathtt{M}, \mathtt{N}, \mathtt{V}, \mathtt{P}$. *In order to avoid unnecessary parenthesis, we use the Barendregt convention, so abstraction associates on the left and applications associates on the right. Moreover* $\lambda\mathtt{xy}.\mathtt{M}$ *stands for* $\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{M}$. *As usual terms are considered up to* α*-equivalence, namely a bound variable can be renamed provided no free variable is captured. Moreover,* $\mathtt{M}[\mathtt{N}/\mathtt{x}]$ *denotes the capture-free substitution of all free occurrences of* $\mathtt{x}$ *in* $\mathtt{M}$ *by* $\mathtt{N}$*,* $\mathrm{FV}(\mathtt{M})$ *denotes the set of free variables of* $\mathtt{M}$ *and* $n_o(\mathtt{x}, \mathtt{M})$ *denotes the number of free occurrences of the variable* $\mathtt{x}$ *in* $\mathtt{M}$.

In the sequel we will be interested only in typable terms.

Definition 2.2 ($\mathrm{STA}_\mathbf{B}$).

(1) The set $\mathcal{T}_\mathbf{B}$ of *types* contains all the objects defined by the following grammar:

$$\begin{array}{ll} A ::= \mathbf{B} \mid \alpha \mid \sigma \multimap A \mid \forall\alpha.A & \text{(Linear Types)} \\ \sigma ::= A \ \mid !\sigma & \text{(Types)} \end{array}$$

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

(2) A *context* is a set of assumptions of the shape $\mathtt{x} : \sigma$, where all variables are different. We use Γ, Δ to denote contexts.

(3) The system $\mathrm{STA}_\mathbf{B}$ proves judgments of the shape $\Gamma \vdash \mathtt{M} : \sigma$ where Γ is a context, $\mathtt{M}$ is a term, and σ is a type. The rules are given in Table I.

NOTATION 2. *Type variables are denoted by* α, β*, linear types by* A, B, C*, and types by* σ, τ, μ. *The symbol* $\equiv$ *denotes the syntactical equality both for types and terms (modulo renaming of bound variables). As usual* $\multimap$ *associates to the right and has precedence on* $\forall$*, while* $!$ *has precedence on everything else. The notation* $\sigma[A/\alpha]$ *stands for the usual capture free substitution in* σ *of all occurrences of the type variable* α *by the linear type* A. *We use* $\mathrm{dom}(\Gamma)$ *and* $\mathrm{FTV}(\Gamma)$ *to denote respectively the sets of variables and of free type variables that occur in the assumptions of the context* Γ. *The notation* $\Gamma\#\Delta$ *stands for* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$. *Derivations are denoted by* Π, Σ, Θ. $\Pi \triangleright \Gamma \vdash \mathtt{M} : \sigma$ *denotes a derivation* Π *with conclusion* $\Gamma \vdash \mathtt{M} : \sigma$. *We let* $\vdash \mathtt{M} : \sigma$ *abbreviate* $\emptyset \vdash \mathtt{M} : \sigma$. *As usual,* $\forall\vec{\alpha}.A$ *is an abbreviation for* $\forall\alpha_1 .... \forall\alpha_m.A$*, and* $!^n\sigma$ *is an abbreviation for* $!...!\sigma$ n*-times* ($m, n \geq 0$).

We stress that each type is of the shape $!^n\forall\vec{\alpha}.A$. The type assignment system $\mathrm{STA}_\mathbf{B}$ is obtained form STA just by adding the rules for dealing with the $\mathtt{if}$ constructor. Note that the rule ($\mathbf{B}E$) has an additive treatment of the contexts, and so contraction is free, while all other rules are multiplicative. Moreover $\mathrm{STA}_\mathbf{B}$ is affine, since the weakening is free, so it enjoys the following properties.

LEMMA 2.3 (FREE VARIABLE LEMMA).

(*1*) $\Gamma \vdash \mathtt{M} : \sigma$ *implies* $\mathrm{FV}(\mathtt{M}) \subseteq \mathrm{dom}(\Gamma)$.
(*2*) $\Gamma \vdash \mathtt{M} : \sigma, \Delta \subseteq \Gamma$ *and* $\mathrm{FV}(\mathtt{M}) \subseteq \mathrm{dom}(\Delta)$ *imply* $\Delta \vdash \mathtt{M} : \sigma$.
(*3*) $\Gamma \vdash \mathtt{M} : \sigma, \Gamma \subseteq \Delta$ *implies* $\Delta \vdash \mathtt{M} : \sigma$.

PROOF. All the three points can be easily proved by induction on the derivation proving $\Gamma \vdash \mathtt{M} : \sigma$. $\square$

Table I. The Soft Type Assignment system with Booleans.

$$\frac{}{\mathtt{x}:A \vdash \mathtt{x}:A}\ (Ax) \qquad \frac{}{\vdash \mathtt{0}:\mathbf{B}}\ (\mathbf{B}_0 I) \qquad \frac{}{\vdash \mathtt{1}:\mathbf{B}}\ (\mathbf{B}_1 I) \qquad \frac{\Gamma \vdash \mathtt{M}:\sigma}{\Gamma, \mathtt{x}:A \vdash \mathtt{M}:\sigma}\ (w)$$

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

$$\frac{\Gamma, \mathtt{x}_1:\sigma, \dots, \mathtt{x}_n:\sigma \vdash \mathtt{M}:\tau}{\Gamma, \mathtt{x}:!\sigma \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n]:\tau}\ (m) \qquad \frac{\Gamma \vdash \mathtt{M}:\sigma}{!\Gamma \vdash \mathtt{M}:!\sigma}\ (sp) \qquad \frac{\Gamma \vdash \mathtt{M}:\forall\alpha.B}{\Gamma \vdash \mathtt{M}:B[A/\alpha]}\ (\forall E)$$

$$\frac{\Gamma \vdash \mathtt{M}:\mathbf{B} \quad \Gamma \vdash \mathtt{N}_0:A \quad \Gamma \vdash \mathtt{N}_1:A}{\Gamma \vdash \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ :A}\ (\mathbf{B}E) \qquad \frac{\Gamma \vdash \mathtt{M}:A \quad \alpha \notin \mathrm{FTV}(\Gamma)}{\Gamma \vdash \mathtt{M}:\forall\alpha.A}\ (\forall I)$$

Moreover, the following property holds:

LEMMA 2.4. $\Gamma, \mathtt{x}:A \vdash \mathtt{M}:!\sigma$ *implies* $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.

PROOF. Easy, by induction on the derivation proving $\Gamma, \mathtt{x}:A \vdash \mathtt{M}:!\sigma$ noticing that the only way to have a modal conclusion is by using the (sp) rule. □

In what follows, we will need to talk about proofs modulo some simple operations.

Definition 2.5. Let Π and Π' be two derivations in $\mathrm{STA}_{\mathbf{B}}$, proving the same conclusion. Then, $\Pi \rightsquigarrow \Pi'$ denotes the fact that Π' is obtained from Π by commuting and/or deleting some rules and/or by inserting some applications of the rule (w).

The system $\mathrm{STA}_{\mathbf{B}}$ is not syntax directed, but the Generation Lemma shows that we can modify the derivations, using the relation $\rightsquigarrow$ defined above, in order to connect the shape of a term with the shape of its typings.

LEMMA 2.6 (GENERATION LEMMA).

(*1*) $\Pi \rhd \Gamma \vdash \lambda\mathtt{x}.\mathtt{M}:\forall\alpha.A$ *implies there is* Π'*, proving the same conclusion as* Π *and ending with an application of rule* $(\forall I)$*, such that* $\Pi \rightsquigarrow \Pi'$.
(*2*) $\Pi \rhd \Gamma \vdash \lambda\mathtt{x}.\mathtt{M}:\sigma \multimap A$ *implies there is* Π'*, proving the same conclusion as* Π *and ending with an application of rule* $(\multimap I)$*, such that* $\Pi \rightsquigarrow \Pi'$.
(*3*) $\Pi \rhd \Gamma \vdash \mathtt{M}:!\sigma$ *implies there is* Π'*, proving the same conclusion as* Π*, such that* $\Pi \rightsquigarrow \Pi'$ *and* Π' *consists of a subderivation, ending with the rule (sp) proving* $!\Gamma' \vdash \mathtt{M}:!\sigma$*, followed by a sequence of rules (w) and/or (m) dealing with variables not occurring in* $\mathtt{M}$.
(*4*) $\Pi \rhd !\Gamma \vdash \mathtt{M}:!\sigma$ *implies there is* Π'*, proving the same conclusion as* Π *and ending with an application of rule* (sp)*, such that* $\Pi \rightsquigarrow \Pi'$.

PROOF.

(1) By induction on Π. If the last rule of Π is $(\forall I)$ then the conclusion follows immediately. Otherwise consider the case $\lambda\mathtt{y}.\mathtt{M} \equiv \lambda\mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n]$ and Π ends as:

$$\frac{\Sigma \rhd \Gamma, \mathtt{x}_1:\sigma, \dots, \mathtt{x}_n:\sigma \vdash \lambda\mathtt{y}.\mathtt{N}:\forall\alpha.A}{\Gamma, \mathtt{x}:!\sigma \vdash \lambda\mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n]:\forall\alpha.A}\ (m)$$

By induction hypothesis $\Sigma \rightsquigarrow \Sigma'$ ending as:

$$\frac{\Sigma_1 \rhd \Gamma, \mathtt{x}_1:\sigma, \dots, \mathtt{x}_n:\sigma \vdash \lambda\mathtt{y}.\mathtt{N}:A}{\Gamma, \mathtt{x}_1:\sigma, \dots, \mathtt{x}_n:\sigma \vdash \lambda\mathtt{y}.\mathtt{N}:\forall\alpha.A}\ (\forall I)$$

Then, the desired Π' is:

$$\cfrac{\cfrac{\Sigma_1 \rhd \Gamma, \mathtt{x}_1 : \sigma, \dots, \mathtt{x}_n : \sigma \vdash \lambda \mathtt{y}.\mathtt{N} : A}{\Gamma, \mathtt{x} :!\sigma \vdash \lambda \mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n] : A}\ (m)}{\Gamma, \mathtt{x} :!\sigma \vdash \lambda \mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n] : \forall \alpha.A}\ (\forall I)$$

The cases where Π ends either by $(\forall E)$ or (w) rule are easier. The other cases are not possible.

(2) Similar to the proof of the previous point of this lemma.

(3) By induction on Π. In the case the last rule of Π is (sp), the proof is obvious. The case where the last rule of Π is (w) follows directly by induction hypothesis. Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n]$ and the last rule is:

$$\frac{\Sigma \rhd \Delta, \mathtt{x}_1 : \tau, ..., \mathtt{x}_n : \tau \vdash \mathtt{N} :!\sigma}{\Delta, \mathtt{x} :!\tau \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n] :!\sigma}\ (m)$$

In the case $\mathtt{x}_1, \dots, \mathtt{x}_n \notin \mathrm{FV}(\mathtt{N})$ the conclusion follows immediately. Otherwise, by induction hypothesis $\Sigma \rightsquigarrow \Sigma_1$, where Σ_1 is composed by a subderivation Θ ending with a rule (sp) proving $!\Delta_1 \vdash \mathtt{N} :!\sigma$, followed by a sequence δ of rules (w) or (m), dealing with variables not occurring in $\mathtt{N}$. Note that for each $\mathtt{x}_i$ with $1 \leq i \leq n$ such that $\mathtt{x}_i \in \mathrm{FV}(\mathtt{N})$, necessarily $\mathtt{x}_i : \tau' \in \Delta_1$ and $\tau =!\tau'$. Let Δ_2 be the context $\Delta_1 - \{\mathtt{x}_1 : \tau', \dots, \mathtt{x}_n : \tau'\}$, then the conclusion follows by the derivation:

$$\cfrac{\cfrac{\Delta_2, \mathtt{x}_1 : \tau', \dots, \mathtt{x}_n : \tau' \vdash \mathtt{N} : \sigma}{\Delta_2, \mathtt{x} :!\tau' \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)}{!\Delta_2, \mathtt{x} :!\tau \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \dots, \mathtt{x}/\mathtt{x}_n] :!\sigma}\ (sp)$$

followed by a sequence of rules (w) recovering the context Δ from the context Δ_2. The other cases are not possible.

(4) By induction on Π. In the case the last rule of Π is (sp), the proof is obvious. The only other possible case is when the last rule is (m). Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n]$ and Π ends as follows:

$$\frac{\Sigma \rhd !\Delta, \mathtt{x}_1 : \tau, ..., \mathtt{x}_n : \tau \vdash \mathtt{N} :!\sigma}{!\Delta, \mathtt{x} :!\tau \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n] :!\sigma}\ (m)$$

If $\tau \equiv !\tau'$, by induction hypothesis $\Sigma \rightsquigarrow \Sigma_1$, where Σ_1 ends as:

$$\frac{\Theta \rhd \Delta, \mathtt{x}_1 : \tau', ..., \mathtt{x}_n : \tau' \vdash \mathtt{N} : \sigma}{!\Delta, \mathtt{x}_1 :!\tau', ..., \mathtt{x}_n :!\tau' \vdash \mathtt{N} :!\sigma}\ (sp)$$

So the desired derivation Π' is Θ, followed by a rule (m) and a rule (sp). In the case τ is linear, by Lemma 2.4, $\mathtt{x}_i \notin \mathrm{FV}(\mathtt{N})$ for each $1 \leq i \leq n$. Moreover by the previous point of this lemma, Σ can be rewritten as:

$$\frac{\Sigma_1 \rhd \Delta_1 \vdash \mathtt{N} : \sigma}{!\Delta_1 \vdash \mathtt{N} :!\sigma}\ (sp)$$

followed by a sequence δ of rules, all dealing with variables not occurring in $\mathtt{N}$. So δ needs to contain some rules introducing the variables $\mathtt{x}_1, ..., \mathtt{x}_n$. Let δ' be the sequence of rules obtained from δ by erasing such rules, and inserting a (w) rule introducing the assignment $\mathtt{x} : \tau$. The desired derivation Π' is Σ_1 followed by δ', followed by (sp). $\square$

2.1. Subject reduction

In order to prove subject reduction, we need to prove before that the system enjoys the property of substitution.

LEMMA 2.7 (SUBSTITUTION LEMMA).
Let $\Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ *and* $\Delta \vdash \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$. *Then*

$$\Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$$

PROOF. We prove something stronger. That is, given

$$\Pi \rhd \Gamma, \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_n : \mu_n \vdash \mathtt{M} : \sigma$$

and $\Sigma_1 \rhd \Delta_1 \vdash \mathtt{N}_1 : \mu_1, \ldots, \Sigma_n \rhd \Delta_n \vdash \mathtt{N}_n : \mu_n$ such that $\Gamma \# \Delta_i$ and $\Delta_i \# \Delta_j$ for each $1 \leq i, j \leq n$, we show that there is a derivation Θ such that

$$\Theta \rhd \Gamma, \Delta_1, \ldots, \Delta_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$$

We proceed by induction on the height of Π. The base cases (Ax), $(\mathbf{B}_0 I)$ and $(\mathbf{B}_1 I)$ are trivial. The cases where Π ends either by $(\multimap I), (\forall I)$ or $(\forall E)$ follow directly from the induction hypothesis.
Let Π ends as

$$\frac{\Pi' \rhd \Gamma', \mathtt{x}_1 : \mu'_1, \ldots, \mathtt{x}_n : \mu'_n \vdash \mathtt{M} : \sigma'}{!\Gamma', \mathtt{x}_1 :!\mu'_1, \ldots, \mathtt{x}_n :!\mu'_n \vdash \mathtt{M} :!\sigma'} \ (sp)$$

By Lemma 2.6.3, for each $1 \leq i \leq n$ we have $\Sigma_i \rightsquigarrow \Sigma''_1$ which is composed by a subderivation ending with an (sp) rule with premise $\Sigma'_i \rhd \Delta'_i \vdash \mathtt{N}_i : \mu'_i$ followed by a sequence of rules (w) and/or (m). By induction hypothesis we have a derivation $\Theta' \rhd \Gamma', \Delta'_1, \ldots, \Delta'_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma'$. By applying the rule (sp) and the sequences of (w) and/or (m) rules we obtain a derivation Θ with conclusion

$$\Gamma, \Delta_1, \ldots, \Delta_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$$

Let Π ends by a $(\multimap E)$ rule. Without loss of generality we can consider a case as follows:

$$\frac{\Pi_1 \rhd \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_i : \mu_i, \Gamma_1 \vdash \mathtt{M} : \sigma \multimap A \quad \Pi_2 \rhd \mathtt{x}_{i+1} : \mu_{i+1}, \ldots, \mathtt{x}_n : \mu_n, \Gamma_2 \vdash \mathtt{N} : \sigma}{\Gamma_1, \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_i : \mu_i, \mathtt{x}_{i+1} : \mu_{i+1}, \ldots, \mathtt{x}_n : \mu_n, \Gamma_2 \vdash \mathtt{MN} : A}$$

By the induction hypothesis there are derivations $\Theta_1 \rhd \Gamma_1, \Delta_1, \ldots, \Delta_i \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_i/\mathtt{x}_i] : \sigma \multimap A$ and $\Theta_2 \rhd \Gamma_2, \Delta_{i+1}, \ldots, \Delta_n \vdash \mathtt{N}[\mathtt{N}_{i+1}/\mathtt{x}_{i+1}, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$. By applying a $(\multimap E)$ rule we obtain a derivation Θ with conclusion:

$$\Gamma_1, \Gamma_2, \Delta_1, \ldots, \Delta_i, \Delta_{i+1}, \ldots, \Delta_n \vdash \mathtt{MN}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : A$$

Consider the case Π ends by:

$$\frac{\Pi_0 \rhd \Gamma' \vdash \mathtt{M}_0 : \mathbf{B} \quad \Pi_1 \rhd \Gamma' \vdash \mathtt{M}_1 : A \quad \Pi_2 \rhd \Gamma' \vdash \mathtt{M}_2 : A}{\Gamma' \vdash \mathtt{if}\ \mathtt{M}_0\ \mathtt{then}\ \mathtt{M}_1\ \mathtt{else}\ \mathtt{M}_2 : A} \ (\mathbf{B}E)$$

with $\Gamma' = \Gamma, \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_n : \mu_n$. Then, by the induction hypothesis there are derivations $\Theta_0 \rhd \Gamma, \Delta_1, \ldots, \Delta_n \vdash \mathtt{M}_0[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \mathbf{B}$, $\Theta_1 \rhd \Gamma, \Delta_1, \ldots, \Delta_n \vdash \mathtt{M}_1[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : A$ and $\Theta_2 \rhd \Gamma, \Delta_1, \ldots, \Delta_n \vdash \mathtt{M}_2[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : A$. By applying a $(\mathbf{B}E)$ rule we obtain a derivation Θ with conclusion:

$$\Gamma, \Delta_1, \ldots, \Delta_n \vdash (\ \mathtt{if}\ \mathtt{M}_0\ \mathtt{then}\ \mathtt{M}_1\ \mathtt{else}\ \mathtt{M}_2\ )[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_n/\mathtt{x}_n] : A$$

Consider the case Π ends by an (m) rule. Note that if the variable $\mathtt{x}$ on which the multiplexing applies is not between $\mathtt{x}_1, \ldots, \mathtt{x}_n$, then the conclusion follows directly by induction hypothesis. So consider a case as follows:

$$\frac{\Pi' \rhd \Gamma, \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_i^1 : \mu'_i, \ldots, \mathtt{x}_i^m : \mu'_i, \ldots, \mathtt{x}_n : \mu_n \vdash \mathtt{M} : \sigma}{\Gamma, \mathtt{x}_1 : \mu_1, \ldots, \mathtt{x}_i :!\mu'_i, \ldots, \mathtt{x}_n : \mu_n \vdash \mathtt{M}[\mathtt{x}_i/\mathtt{x}_i^1, \ldots, \mathtt{x}_i/\mathtt{x}_i^m] : \sigma} \ (m)$$

By Lemma 2.6.3 $\Sigma_i \rightsquigarrow \Sigma_i''$ ending by an (sp) rule with premise $\Sigma_i' \triangleright \Delta_i' \vdash \mathtt{N}_i : \mu_i'$ followed by a sequence of rules (w) and/or (m). Consider fresh copies of the derivation Σ_i' i.e. $\Sigma_i^j \triangleright \Delta_i^j \vdash \mathtt{N}_i^j : \mu_i'$ where $\mathtt{N}_i^j$ and Δ_i^j are fresh copies of $\mathtt{N}_i$ and Δ_i' $(1 \leq j \leq m)$. Then, by induction hypothesis we have a derivation Θ' proving

$$\Gamma, \Delta_1, \ldots, \Delta_i^1, \ldots, \Delta_i^m, \ldots, \Delta_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_i^1/\mathtt{x}_i^1, \ldots, \mathtt{N}_i^m/\mathtt{x}_i^m, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$$

By applying a sequence of (m) rules we can obtain a derivation Θ'' proving

$$\Gamma, \Delta_1, \ldots, !\Delta', \ldots, \Delta_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_i/\mathtt{x}_i^1, \ldots, \mathtt{N}_i/\mathtt{x}_i^m, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$$

Finally by applying the sequence of rules (m) and (w) rules we obtain a derivation Θ with conclusion:

$$\Gamma, \Delta_1, \ldots, \Delta_i, \ldots, \Delta_n \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \ldots, \mathtt{N}_i/\mathtt{x}_i^1, \ldots, \mathtt{N}_i/\mathtt{x}_i^m, \ldots, \mathtt{N}_n/\mathtt{x}_n] : \sigma$$

This concludes the proof. □

We can finally prove the main property of this section.

LEMMA 2.8 (SUBJECT REDUCTION). *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$. *Then,* $\Gamma \vdash \mathtt{N} : \sigma$.

PROOF. By induction on the derivation $\Theta \triangleright \Gamma \vdash \mathtt{M} : \sigma$. Consider the case of a $\to_\delta$ reduction. Without loss of generality we can consider only the case Θ ends as:

$$\frac{\Pi \triangleright \Gamma \vdash \mathtt{b} : \mathbf{B} \quad \Pi_0 \triangleright \Gamma \vdash \mathtt{M}_0 : A \quad \Pi_1 \triangleright \Gamma \vdash \mathtt{M}_1 : A}{\Gamma \vdash \ \mathtt{if\ b\ then\ M_0\ else\ M_1} \ : A} \ (\mathbf{B}E)$$

where $\mathtt{b}$ is either $\mathtt{0}$ or $\mathtt{1}$. The others follow directly by induction hypothesis. If $\mathtt{b} \equiv \mathtt{0}$ then $\mathtt{if\ b\ then\ M_0\ else\ M_1} \ \to_\delta \mathtt{M}_0$ and since $\Pi_0 \triangleright \Gamma \vdash \mathtt{M}_0 : A$, the conclusion follows. Analogously if $\mathtt{b} \equiv \mathtt{1}$ then $\mathtt{if\ b\ then\ M_0\ else\ M_1} \ \to_\delta \mathtt{M}_1$ and since $\Pi_1 \triangleright \Gamma \vdash \mathtt{M}_1 : A$, the conclusion follows.
Now consider the case of a $\to_\beta$ reduction. Without loss of generality we can consider only the case Θ ends as:

$$\frac{\Pi \triangleright \Gamma_1 \vdash \lambda\mathtt{x}.\mathtt{M} : \sigma \multimap A \quad \Sigma \triangleright \Gamma_2 \vdash \mathtt{N} : \sigma}{\Gamma_1, \Gamma_2 \vdash (\lambda\mathtt{x}.\mathtt{M})\mathtt{N} : A} \ (\multimap E)$$

where $\Gamma = \Gamma_1, \Gamma_2$. The others follow directly by induction hypothesis. Clearly $(\lambda\mathtt{x}.\mathtt{M})\mathtt{N} \to_\beta \mathtt{M}[\mathtt{N}/\mathtt{x}]$. By Lemma 2.6.2 $\Pi \rightsquigarrow \Pi_1$ ending as

$$\frac{\Pi_2 \triangleright \Gamma_1, \mathtt{x} : \sigma \vdash \mathtt{M} : A}{\Gamma_1 \vdash \lambda\mathtt{x}.\mathtt{M} : \sigma \multimap A}$$

By the Substitution Lemma 2.7 since $\Pi_2 \triangleright \Gamma_1, \mathtt{x} : \sigma \vdash \mathtt{M} : A$ and $\Sigma \triangleright \Gamma_2 \vdash \mathtt{N} : \sigma$ we have $\Gamma_1, \Gamma_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : A$, hence the conclusion follows. □

It is worth noticing that, due to the additive rule $(\mathbf{B}E)$, $\mathrm{STA}_\mathbf{B}$ is no more correct for polynomial time, since terms with exponential number of reductions can be typed by derivations with a priori fixed degree, where the degree is the nesting of (sp) applications.

Example 2.9. Consider for $n \in \mathbb{N}$ terms $\mathtt{M}_n$ of the shape:

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

It is easy to verify that for each $\mathtt{M}_n$ there are reduction sequences of length exponential in n.

Table II. System F with explicit contraction and weakening rules.

$$\frac{}{\mathtt{x}:C \vdash_F \mathtt{x}:C}\ (Ax) \qquad \frac{\Gamma \vdash_F \mathtt{M}:B}{\Gamma, \mathtt{x}:C \vdash_F \mathtt{M}:B}\ (w) \qquad \frac{\Gamma, \mathtt{x}_1:C, \mathtt{x}_2:C \vdash_F \mathtt{M}:B}{\Gamma, \mathtt{x}:C \vdash_F \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \mathtt{x}/\mathtt{x}_2]:B}\ (c)$$

$$\frac{\Gamma, \mathtt{x}:C \vdash_F \mathtt{M}:B}{\Gamma \vdash_F \lambda\mathtt{x}.\mathtt{M}:C \Rightarrow B}\ (\Rightarrow I) \qquad \frac{\Gamma \vdash_F \mathtt{M}:C \Rightarrow B \quad \Delta \vdash_F \mathtt{N}:C}{\Gamma, \Delta \vdash_F \mathtt{MN}:B}\ (\Rightarrow E)$$

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

2.2. Strong Normalization

Strong normalization is proved by a translation, preserving reduction, of $\mathrm{STA}_{\mathbf{B}}$ in a slightly variant of Girard's System F [Girard 1972]. The variant we consider is showed in Table II and it differs from the original system since it has explicit rules for weakening and contraction. It is straightforward to prove that it shares all the properties of the original one, in particular strong normalization.

Definition 2.10. The set $\mathcal{T}_F$ of *System F* types is defined by the grammar:

$$B, C \ ::= \ \alpha \mid B \Rightarrow B \mid \forall\alpha.B$$

where α ranges over a countable set of type variables.

We firstly define a forgetful map over types and terms.

Definition 2.11. The map $(-)^* : \mathcal{T}_{\mathbf{B}} \cup \Lambda_{\mathcal{B}} \to \mathcal{T}_F \cup \Lambda$ is defined on types as:

$$(\mathbf{B})^* = \forall\alpha.\alpha \Rightarrow \alpha \Rightarrow \alpha \qquad (\alpha)^* = \alpha \qquad (\sigma \multimap A)^* = (\sigma)^* \Rightarrow (A)^*$$
$$(!\sigma)^* = (\sigma)^* \qquad (\forall\alpha.A)^* = \forall\alpha.(A)^*$$

and it is defined on terms as:

$$(\mathtt{0})^* = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x} \qquad (\mathtt{1})^* = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y} \qquad (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{M}_1\ \mathtt{else}\ \mathtt{M}_2\ )^* = (\mathtt{M})^*(\mathtt{M}_1)^*(\mathtt{M}_2)^*$$
$$(\lambda\mathtt{x}.\mathtt{M})^* = \lambda\mathtt{x}.(\mathtt{M})^* \qquad (\mathtt{MN})^* = (\mathtt{M})^*(\mathtt{N})^*$$

The following lemma assures that the translation well behaves.

LEMMA 2.12. *If* $\Gamma \vdash \mathtt{M}:\sigma$ *then* $(\Gamma)^* \vdash_F (\mathtt{M})^* : (\sigma)^*$.

PROOF. By induction on the derivation Π proving $\Gamma \vdash \mathtt{M}:\sigma$.
Let us consider base cases. The (Ax) case is trivial. Consider the case Π consists in the rule

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

Then we have the following derivation

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\mathtt{x}:\alpha \vdash_F \mathtt{x}:\alpha}\ (Ax)}{\mathtt{y}:\alpha, \mathtt{x}:\alpha \vdash_F \mathtt{x}:\alpha}\ (w)}{\mathtt{x}:\alpha \vdash_F \lambda\mathtt{y}.\mathtt{x}:\alpha \Rightarrow \alpha}\ (\Rightarrow I)}{\vdash_F \lambda\mathtt{xy}.\mathtt{x}:\alpha \Rightarrow \alpha \Rightarrow \alpha}\ (\Rightarrow I)}{\vdash_F \lambda\mathtt{xy}.\mathtt{x}:\forall\alpha.\alpha \Rightarrow \alpha \Rightarrow \alpha}\ (\forall I)$$

The case Π consists in the $(\mathbf{B}_1 I)$ rule is similar. The case Π ends by (sp) rule follows directly from the induction hypothesis. In case Π ends by a $(\multimap I), (\multimap E), (w)$ rule the conclusion follows from induction hypothesis and an application of the same rule in System F. In the case Π ends by a (m) rule the conclusion follows from induction hypothesis and some applications of the (c) rule. In the case Π ends as

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N}_0 : A \quad \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A}\ (\mathbf{B}E)$$

by induction hypothesis we have derivations $\Pi_1 \rhd (\Gamma)^* \vdash_F (\mathtt{M})^* : (\mathbf{B})^*$, $\Pi_2 \rhd (\Gamma)^* \vdash_F (\mathtt{N}_0)^* : (A)^*$ and $\Pi_3 \rhd (\Gamma)^* \vdash_F (\mathtt{N}_1)^* : (A)^*$. Let Π'_1, Π'_2 and Π'_3 be copies of Π_1, Π_2 and Π_3 respectively with different renaming of free variables. That is, $\Pi'_1 \rhd \Gamma_1 \vdash_F \mathtt{M}' : (\mathbf{B})^*$ $\Pi'_2 \rhd \Gamma_2 \vdash_F \mathtt{N}'_0 : (A)^*$ and $\Pi'_3 \rhd \Gamma_3 \vdash_F \mathtt{N}'_1 : (A)^*$. So, we can build a derivation ending as:

$$\cfrac{\cfrac{\cfrac{\Gamma_1 \vdash_F \mathtt{M}' : (\mathbf{B})^* = \forall\alpha.\alpha \Rightarrow \alpha \Rightarrow \alpha}{\Gamma_1 \vdash_F \mathtt{M}' : (A)^* \Rightarrow (A)^* \Rightarrow (A)^*} \quad \Gamma_2 \vdash_F \mathtt{N}'_0 : (A)^*}{\Gamma_1, \Gamma_2 \vdash_F \mathtt{M}'\mathtt{N}'_0 : (A)^* \Rightarrow (A)^*} \qquad \Gamma_3 \vdash_F \mathtt{N}'_1 : (A)^*}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash_F \mathtt{M}'\mathtt{N}'_0\mathtt{N}'_1 : (A)^*}$$

So, the conclusion $\Gamma \vdash (\mathtt{M})^*(\mathtt{N}_0)^*(\mathtt{N}_1)^* : (A)^*$ follows from several applications of the rule (c). □

Moreover, the translation preserves the reduction.

LEMMA 2.13 (SIMULATION). *If* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$ *then* $(\mathtt{M})^* \to^+_\beta (\mathtt{N})^*$.

PROOF. The case of a β-reduction is trivial, so consider a δ-reduction as:

$$\mathtt{M} = \mathtt{C}[\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{Q}\ ] \to_\delta \mathtt{C}[\mathtt{P}] = \mathtt{N}$$

By definition of the map $(\ )^*$ we have:

$$(\mathtt{M})^* = (\mathtt{C}[\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{P}\ \mathtt{else}\ \mathtt{Q}\ ])^* = \mathtt{C}'[(\mathtt{0})^*(\mathtt{P})^*(\mathtt{Q})^*] = \mathtt{C}'[(\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{x})(\mathtt{P})^*(\mathtt{Q})^*]$$

and clearly:

$$\mathtt{C}'[(\lambda \mathrm{x}.\lambda \mathrm{y}.\mathrm{x})(\mathtt{P})^*(\mathtt{Q})^*] \to_\beta \mathtt{C}'[(\lambda \mathrm{y}.(\mathtt{P})^*)(\mathtt{Q})^*] \to_\beta \mathtt{C}'[(\mathtt{P})^*] = (\mathtt{N})^*$$

and so the conclusion. The other case is analogous. □

Now, we have the following.

THEOREM 2.14 (STRONG NORMALIZATION).
If $\Gamma \vdash \mathtt{M} : \sigma$ *then* $\mathtt{M}$ *is strongly normalizing with respect to the relation* $\to_{\beta\delta}$.

PROOF. By Lemmas 2.12 and 2.13 and the strong normalization of System F. □

3. STRUCTURAL OPERATIONAL SEMANTICS

In this section the operational semantics of terms of $\Lambda_\mathcal{B}$ is presented, through an evaluation machine, named $\mathrm{K}^\mathcal{C}_\mathcal{B}$, defined in SOS style [Plotkin 2004; Kahn 1987]. The machine $\mathrm{K}^\mathcal{C}_\mathcal{B}$ is related to the type assignment system $\mathrm{STA}_\mathbf{B}$ since it evaluates programs (i.e., closed terms of boolean type). The machine allows us to measure the space used during the evaluation. In order to justify our space measure, a small step version of $\mathrm{K}^\mathcal{C}_\mathcal{B}$ is used.

3.1. The evaluation machine $\mathrm{K}^\mathcal{C}_\mathcal{B}$

The machine $\mathrm{K}^\mathcal{C}_\mathcal{B}$ evaluates programs according to the leftmost outermost strategy. If restricted to λ-calculus, the machine $\mathrm{K}^\mathcal{C}_\mathcal{B}$ is quite similar to the Krivine machine [Krivine 2007], since β-reduction is not an elementary step, but the substitution of a term to a variable is performed one occurrence at a time. The machine $\mathrm{K}^\mathcal{C}_\mathcal{B}$ uses two memory devices, the m-context and the $\mathbf{B}$-context, that memorize respectively the assignments to variables and the control flow.

Definition 3.1.

— An *m-context* $\mathcal{A}$ is a sequence of variable assignments of the shape $\mathtt{x} := \mathtt{M}$ where $\mathtt{M}$ is a term and all the variables are distinct. The symbol ε denotes the empty m-context and the set of m-contexts is denoted by $\mathrm{Ctx}_{\mathrm{m}}$.
The *cardinality* of an m-context $\mathcal{A}$, denoted by $\#(\mathcal{A})$, is the number of variable assignments in $\mathcal{A}$. The *size* of an m-context $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the sum of the sizes of all the variable assignments in $\mathcal{A}$, where a variable assignment $\mathtt{x} := \mathtt{M}$ has size $|\mathtt{M}| + 1$.
— Let $\circ$ be a distinguished symbol. The set $\mathrm{Ctx}_{\mathbf{B}}$ of $\mathbf{B}$-*contexts* is defined by the following grammar:

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

The size of a $\mathbf{B}$-context $\mathcal{C}[\circ]$, denoted by $|\mathcal{C}[\circ]|$, is the size of the term obtained by replacing the symbol $\circ$ by a variable.
The cardinality of a $\mathbf{B}$-context $\mathcal{C}[\circ]$, denoted by $\#(\mathcal{C}[\circ])$, is the number of nested $\mathbf{B}$-contexts in it. i.e.:

$$\#(\circ) = 0 \qquad \#((\ \mathtt{if}\ \mathcal{C}[\circ]\ \mathtt{then}\ \mathtt{M}\ \mathtt{else}\ \mathtt{N}\ )\mathtt{V}_1 \cdots \mathtt{V}_n) = \#(\mathcal{C}[\circ]) + 1$$

It is worth noticing that a $\mathbf{B}$-contexts $\mathcal{C}[\circ]$ can be seen as a stack of atomic contexts, i.e. contexts of the shape $(\ \mathtt{if}\ \circ\ \mathtt{then}\ \mathtt{M}\ \mathtt{else}\ \mathtt{N}\ )\mathtt{V}_1 \cdots \mathtt{V}_n$, and that the cardinality $\#(\mathcal{C}[\circ])$ is the height of such a stack.

NOTATION 3. *The notation* $\mathcal{A}_1 @ \mathcal{A}_2$ *is used for the concatenation of the disjoint m-contexts* $\mathcal{A}_1$ *and* $\mathcal{A}_2$. *Moreover,* $[\mathtt{x} := \mathtt{M}] \in \mathcal{A}$ *denotes the fact that* $\mathtt{x} := \mathtt{M}$ *is in the m-context* $\mathcal{A}$. *The notation* $\mathrm{FV}(\mathcal{A})$ *identifies the set:* $\bigcup_{[\mathtt{x}:=\mathtt{M}]\in\mathcal{A}} \mathrm{FV}(\mathtt{M})$.
As usual, $\mathcal{C}[\mathtt{M}]$ *denotes the term obtained by filling the hole* $[\circ]$ *in* $\mathcal{C}[\circ]$ *by* $\mathtt{M}$. *In general we omit the hole* $[\circ]$ *and we range over* $\mathbf{B}$*-contexts by* $\mathcal{C}$. *As expected,* $\mathrm{FV}(\mathcal{C})$ *denotes the set* $\mathrm{FV}(\mathcal{C}[\mathtt{M}])$ *for every closed term* $\mathtt{M}$.

Note that variable assignments in m-contexts are ordered; this fact allows us to define the following closure operation.

Definition 3.2. Let $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n := \mathtt{N}_n]$ be a m-context. Then, $(-)^{\mathcal{A}} : \Lambda_{\mathcal{B}} \to \Lambda_{\mathcal{B}}$ is the map associating to each term $\mathtt{M}$ the term $(\mathtt{M})^{\mathcal{A}} \equiv \mathtt{M}[\mathtt{N}_n/\mathtt{x}_n][\mathtt{N}_{n-1}/\mathtt{x}_{n-1}] \cdots [\mathtt{N}_1/\mathtt{x}_1]$.

The correct inputs for the machine are programs, defined as follows.

Definition 3.3. The set $\mathcal{P}$ of *programs* is the set of closed terms typable by the ground type. i.e. $\mathcal{P} = \{\mathtt{M} \mid \vdash \mathtt{M} : \mathbf{B}\}$.

The design of the evaluation machine follows the syntactic shape of programs.

Remark 3.4. It is easy to check that every term has the following shape: $\lambda \mathtt{x}_1 \ldots \mathtt{x}_n . \zeta \mathtt{V}_1 \cdots \mathtt{V}_m$, for some $n, m \geq 0$, where ζ is either a boolean b, a variable $\mathtt{x}$, a redex $(\lambda \mathtt{x}.\mathtt{N})\mathtt{P}$, or a subterm of the shape $\mathtt{if}\ \mathtt{P}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1$. It is immediate to check that, if a term is in $\mathcal{P}$, then $n = 0$. Moreover, if a term in $\mathcal{P}$ is a normal form, then it coincides with a boolean constant $\mathtt{b}$.

The evaluation machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ proves statements of the shape:

$$\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$$

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

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

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

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

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

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

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

where $\mathcal{C},\mathcal{A}$ are a **B**-context and a m-context respectively, $\mathtt{M}$ is a term, and $\mathtt{b}$ is a boolean value. Its rules are listed in Table III. The (β) rule applies when the head of the subject is a β-redex, then the association between the bound variable and the argument is recorded in the m-context and the body of the term in functional position is evaluated. Note that an α-rule is always performed. The (h) rule replaces the head occurrence of the head variable by the term associated with it in the m-context. Rules (if 0) and (if 1) perform the δ reductions. In order to evaluate the test $\mathtt{M}$, a part of the subject is naturally erased. This erased information is stored in the **B**-context. Indeed **B**-contexts are stacks that permit to store all the branches of a computation produced by conditionals. When the evaluation of the test $\mathtt{M}$ of the current conditional is completed, the machine pops the top **B**-context and continues by evaluating the term in the right branch of the computation. The behaviour of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ is formalized in the following definition.

Definition 3.5.

(1) The *evaluation relation* $\Downarrow \subseteq \mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx}_{\mathrm{m}} \times \Lambda_{\mathcal{B}} \times \mathcal{B}$ is the relation inductively defined by the rules of $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. If $\mathtt{M}$ is a program, and if there is a boolean b such that $\circ, \varepsilon \models \mathtt{M} \Downarrow \mathtt{b}$ then we say that $\mathtt{M}$ *evaluates*, and we write $\mathtt{M} \Downarrow$. As usual, $\models \mathtt{M} \Downarrow \mathtt{b}$ is a short for $\circ, \varepsilon \models \mathtt{M} \Downarrow \mathtt{b}$.
(2) Derivation trees in the abstract machine are called *computations* and are denoted by $\nabla, \diamond$. We use $\nabla :: \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ to denote the computation with conclusion $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$.
(3) Given a computation ∇ each node of ∇, which is of the shape $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ is a *configuration*. The notation $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b} \in \nabla$ is used to stress that $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ is a configuration in the computation ∇. Configurations are denoted by ϕ, ψ. The notation $\phi \succ \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ means that ϕ is the configuration $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$. The conclusion of the derivation tree is called the *initial configuration*.
(4) Given a computation ∇, the *path* to reach a configuration ϕ, denoted $\mathtt{path}_{\nabla}(\phi)$, is the sequence of configurations between the conclusion of ∇ and ϕ. In general, we simply write $\mathtt{path}(\phi)$ when ∇ is clear from the context.

In Table IV we present an example of $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ computation on a term $\mathtt{M}_2$ as defined in Example 2.9.

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

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\dfrac{\dfrac{}{\mathcal{C}_1,\mathcal{A}_3 \models \mathtt{0} \Downarrow \mathtt{0}}}{\mathcal{C}_1,\mathcal{A}_3 \models \mathtt{z}_1 \Downarrow \mathtt{0}}}{\mathcal{C}_1,\mathcal{A}_3 \models \mathtt{x}_2 \Downarrow \mathtt{0}}
\quad
\dfrac{\dfrac{\dfrac{}{\phi \succ \mathcal{C}_0,\mathcal{A}_3 \models \mathtt{0} \Downarrow \mathtt{0}}}{\mathcal{C}_0,\mathcal{A}_3 \models \mathtt{z}_1 \Downarrow \mathtt{0}}}{\mathcal{C}_0,\mathcal{A}_3 \models \mathtt{x}_2 \Downarrow \mathtt{0}}
}{\mathcal{C}_0,\mathcal{A}_3 \models \ \mathtt{if}\ \mathtt{x}_2\ \mathtt{then}\ \mathtt{x}_2\ \mathtt{else}\ \mathtt{x}_2 \Downarrow \mathtt{0}}
}{\mathcal{C}_0,\mathcal{A}_2 \models (\lambda\mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x})\mathtt{z}_1 \Downarrow \mathtt{0}}
}{\mathcal{C}_0,\mathcal{A}_2 \models \mathtt{f}_1\mathtt{z}_1 \Downarrow \mathtt{0}}
}{\mathcal{C}_0,\mathcal{A}_2 \models \mathtt{x}_1 \Downarrow \mathtt{0}}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\dfrac{\dfrac{}{\mathcal{C}_2,\mathcal{A}_4 \models \mathtt{0} \Downarrow \mathtt{0}}}{\mathcal{C}_2,\mathcal{A}_4 \models \mathtt{z}_1 \Downarrow \mathtt{0}}}{\mathcal{C}_2,\mathcal{A}_4 \models \mathtt{x}_3 \Downarrow \mathtt{0}}
\quad
\dfrac{\dfrac{\dfrac{}{\mathcal{A}_4 \models \mathtt{0} \Downarrow \mathtt{0}}}{\mathcal{A}_4 \models \mathtt{z}_1 \Downarrow \mathtt{0}}}{\mathcal{A}_4 \models \mathtt{x}_3 \Downarrow \mathtt{0}}
}{\mathcal{A}_4 \models \ \mathtt{if}\ \mathtt{x}_3\ \mathtt{then}\ \mathtt{x}_3\ \mathtt{else}\ \mathtt{x}_3 \Downarrow \mathtt{0}}
}{\mathcal{A}_2 \models (\lambda\mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x})\mathtt{z}_1 \Downarrow \mathtt{0}}
}{\mathcal{A}_2 \models \mathtt{f}_1\mathtt{z}_1 \Downarrow \mathtt{0}}
}{\psi \succ \mathcal{A}_2 \models \mathtt{x}_1 \Downarrow \mathtt{0}}
}{
\dfrac{
\dfrac{
\dfrac{
\mathcal{A}_2 \models \ \mathtt{if}\ \mathtt{x}_1\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1 \Downarrow \mathtt{0}
}{\mathcal{A}_1 \models (\lambda\mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x})(\mathtt{f}_1\mathtt{z}_1) \Downarrow \mathtt{0}}
}{\mathcal{A}_1 \models \mathtt{f}_1(\mathtt{f}_1\mathtt{z}_1) \Downarrow \mathtt{0}}
}{
\dfrac{\mathcal{A}_0 \models (\lambda z.\mathtt{f}_1(\mathtt{f}_1\mathtt{z}))\mathtt{0} \Downarrow \mathtt{0}}{\models (\lambda\mathtt{f}.\lambda\mathtt{z}.\mathtt{f}^2\mathtt{z})(\lambda\mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x})\mathtt{0} \Downarrow \mathtt{0}}
}
}
$$

$$
\begin{array}{ll}
\mathcal{A}_0 = [\mathtt{f}_1 := \lambda\mathtt{x}.\ \mathtt{if\ x\ then\ x\ else\ x}] & \mathcal{C}_0 = \ \mathtt{if}\ \circ\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1 \\
\mathcal{A}_1 = \mathcal{A}_0 @ [\mathtt{z}_1 := \mathtt{0}] & \mathcal{C}_1 = \mathcal{C}_0[\ \mathtt{if}\ \circ\ \mathtt{then}\ \mathtt{x}_2\ \mathtt{else}\ \mathtt{x}_2\ ] \\
\mathcal{A}_2 = \mathcal{A}_1 @ [\mathtt{x}_1 := \mathtt{f}_1\mathtt{z}_1] & \mathcal{C}_2 = \ \mathtt{if}\ \circ\ \mathtt{then}\ \mathtt{x}_3\ \mathtt{else}\ \mathtt{x}_3 \\
\mathcal{A}_3 = \mathcal{A}_2 @ [\mathtt{x}_2 := \mathtt{z}_1] & \\
\mathcal{A}_4 = \mathcal{A}_2 @ [\mathtt{x}_3 := \mathtt{z}_1] &
\end{array}
$$

In order to prove that the machine is sound and complete with respect to programs, we need to prove some additional properties. First of all, the next lemma proves that the machine enjoys a sort of weakening, with respect to both contexts.

LEMMA 3.6.

(*1*) *Let* $\mathcal{C}[\circ], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$. *Then, for every* $\mathcal{C}'[\circ]$ *such that* $(\mathcal{C}'[\mathcal{C}[\mathtt{M}]])^{\mathcal{A}} \in \mathcal{P}$, $\mathcal{C}'[\mathcal{C}[\circ]], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$.
(*2*) *Let* $\nabla :: \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *and* $\mathtt{x}$ *be a fresh variable. Then,* $\nabla :: \mathcal{C}, \mathcal{A}@[\mathtt{x} := \mathtt{N}] \models \mathtt{M} \Downarrow \mathtt{b}$.

PROOF. Both points can be easily proved by induction on the computation. □

LEMMA 3.7.

(*1*) *Let* $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *for some* $\mathtt{b}$ *and let* $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \in \mathcal{P}$. *Then, both* $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$ *and* $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$*, for some* $\mathtt{b}'$.
(*2*) *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *For each* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$, $(\mathcal{C}[\mathtt{N}])^{\mathcal{A}} \in \mathcal{P}$.
(*3*) *Let* $(\mathtt{M})^{\mathcal{A}} \in \mathcal{P}$ *and* $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$. *Then,* $\circ, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$.

PROOF.

(1) First of all, the property $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$, for some $\mathtt{b}'$ derives directly from the fact that $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \in \mathcal{P}$. In fact this implies $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}}$ is a closed strongly normalizing term of type **B**, and so its normal form is necessarily a boolean constant. So in what follows we will prove just that $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ and $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \in \mathcal{P}$ implies $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$. Note that if $(\mathcal{C}[\mathtt{M}])^{\mathcal{A}} \in \mathcal{P}$ then clearly $(\mathtt{M})^{\mathcal{A}} \in \mathcal{P}$. We proceed by induction on the derivation proving $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$. Let the last rule be:

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

Obviously $(\mathtt{b})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$. Let the derivation ends as:

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

By induction hypothesis $(\mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{N}]} \to^*_{\beta\delta} \mathtt{b}$. Clearly since $\mathtt{x}'$ is fresh:

$$(\mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{N}]} \equiv ((\mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)[\mathtt{N}/\mathtt{x}'])^{\mathcal{A}} \equiv (\mathtt{P}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$

hence:

$$((\lambda \mathtt{x}.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to_{\beta\delta} (\mathtt{P}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$$

and the conclusion follows. The case of a rule (h) follows directly by induction hypothesis.
Let the derivation end as:

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

where $\mathcal{C}' = \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$. By induction hypothesis $(\mathtt{P})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{0}$, hence:

$$((\ \mathtt{if}\ \mathtt{P}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} ((\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$

and by δ reduction

$$((\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to_{\delta} (\mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$

moreover, since by induction hypothesis we also have $(\mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$, the conclusion follows. The case of rule $(\ \mathtt{if}\ \ \mathtt{1})$ is analogous.

(2) Easy, by induction on the length of $\mathtt{path}(\phi)$.

(3) The proof is by induction on the number of steps needed to reach the normal form $\mathtt{b}$ of $(\mathtt{M})^{\mathcal{A}}$ according to the leftmost strategy. Since $(\mathtt{M})^{\mathcal{A}}$ is strongly normalizing this is clearly well-founded.
If $(\mathtt{M})^{\mathcal{A}}$ is already in normal form, since it must be typable of type $\mathbf{B}$ then $\mathtt{M} \equiv \mathtt{b}$, and the result is trivial. Otherwise $(\mathtt{M})^{\mathcal{A}}$ cannot be an abstraction, since its typing, so it is an application $\mathtt{N}\mathtt{Q}\mathtt{V}_1...\mathtt{V}_m$.
Suppose $\mathtt{N} \equiv \lambda\mathtt{x}.\mathtt{R}$. There are two cases, either $(\mathtt{M})^{\mathcal{A}} \equiv ((\lambda \mathtt{x}.\mathtt{R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}}$ or $(\mathtt{M})^{\mathcal{A}} \equiv (\mathtt{y}\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}}$ and $[\mathtt{y} := \lambda \mathtt{x}.\mathtt{R}'] \in \mathcal{A}$.
Let us consider the first case. Then $((\lambda \mathtt{x}.\mathtt{R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \to_{\beta} (\mathtt{R}'[\mathtt{Q}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{Q}']}$. By induction hypothesis we have $[\circ], \mathcal{A}@[\mathtt{x}' := \mathtt{Q}'] \models \mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$ and so the result follows by rule (β).
In the second case, since $[\mathtt{y} := \lambda \mathtt{x}.\mathtt{R}'] \in \mathcal{A}$, then $(\mathtt{y}\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \to_{\beta} (\mathtt{R}'[\mathtt{Q}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{Q}']}$. By induction hypothesis $[\circ], \mathcal{A}@[\mathtt{x}' := \mathtt{Q}'] \models \mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$, so by one application of the rule (β), $[\circ], \mathcal{A} \models (\lambda \mathtt{x}.\mathtt{R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$. Finally, by one application of the rule (h), since $[\mathtt{y} := \lambda \mathtt{x}.\mathtt{R}'] \in \mathcal{A}$, we have $[\circ], \mathcal{A} \models \mathtt{y}\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$.
The remaining case is the one where $\mathtt{N} \equiv\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}'_0\ \mathtt{else}\ \mathtt{N}'_1\ $. By definition $(\mathtt{M})^{\mathcal{A}} \equiv ((\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}'_0\ \mathtt{else}\ \mathtt{N}'_1\ )\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\ \mathtt{if}\ (\mathtt{M}')^{\mathcal{A}}\ \mathtt{then}\ (\mathtt{N}'_0)^{\mathcal{A}}\ \mathtt{else}\ (\mathtt{N}'_1)^{\mathcal{A}}\ )(\mathtt{Q}')^{\mathcal{A}}(\mathtt{V}'_1)^{\mathcal{A}}...(\mathtt{V}'_m)^{\mathcal{A}}$. Since $(\mathtt{M})^{\mathcal{A}} \in \mathcal{P}$, $\vdash (\mathtt{M})^{\mathcal{A}} : \mathbf{B}$, so, by the strong normalization property, $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$. This implies either $(\mathtt{M}')^{\mathcal{A}} = \mathtt{b}'$ or $(\mathtt{M}')^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$ for some $\mathtt{b}'$. Let us consider the latter case. The number of reduction steps of the sequence $(\mathtt{M}')^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$ is shorter than the one of $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$, so by induction $[\circ], \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{b}'$, and, by Lemma 3.6.1, $(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}'_0\ \mathtt{else}\ \mathtt{N}'_1\ )\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m, \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{b}'$. Without loss of generality, we consider only the case where $\mathtt{b}' = \mathtt{0}$. Then $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$ implies $(\mathtt{N}'_0\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$, so

Table V. (a) The small step machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$. (b) The garbage collector procedure.

$$\frac{\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}@[\mathtt{x}' := \mathtt{N}], \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)}{\langle \mathcal{C}, \mathcal{A} \succ (\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \rangle} \ (\beta_0)^{\S}$$

$$\frac{[\mathtt{x} := \mathtt{N}] \in \mathcal{A} \quad \mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)}{\langle \mathcal{C}, \mathcal{A} \succ \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle} \ (h_0)$$

$$\frac{\mathcal{C}' = \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n]}{\langle \mathcal{C}, \mathcal{A} \succ (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}', \mathcal{A} \succ \mathtt{M} \rangle} \ (\ \mathtt{if}\ )$$

$$\frac{\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_n)}{\langle \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n], \mathcal{A} \succ \mathtt{0} \rangle \mapsto \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_n \rangle} \ (r_0)$$

$$\frac{\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_n)}{\langle \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n], \mathcal{A} \succ \mathtt{1} \rangle \mapsto \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_n \rangle} \ (r_1)$$

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

$$\frac{}{\mathtt{clear}(\mathcal{C}, \varepsilon, \mathtt{M}) = \varepsilon}$$

$$\frac{\mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{M}) = \mathcal{A}' \quad \mathtt{x} \in \mathrm{FV}(\mathcal{C}) \cup \mathrm{FV}(\mathtt{M}) \cup \mathrm{FV}(\mathcal{A})}{\mathtt{clear}(\mathcal{C}, [\mathtt{x} := \mathtt{N}]@\mathcal{A}, \mathtt{M}) = [\mathtt{x} := \mathtt{N}]@\mathcal{A}'}$$

$$\frac{\mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{M}) = \mathcal{A}' \quad \mathtt{x} \notin \mathrm{FV}(\mathcal{C}) \cup \mathrm{FV}(\mathtt{M}) \cup \mathrm{FV}(\mathcal{A})}{\mathtt{clear}(\mathcal{C}, [\mathtt{x} := \mathtt{N}]@\mathcal{A}, \mathtt{M}) = \mathcal{A}'}$$

by induction $[\circ], \mathcal{A} \models \mathtt{N}'_0\mathtt{Q}'\mathtt{V}'_1 ... \mathtt{V}'_m \Downarrow \mathtt{b}$, and the result follows by rule ($\mathtt{if\ 0}$). The case $(\mathtt{M}')^{\mathcal{A}} = \mathtt{b}'$ is easier. The case $(\mathtt{M})^{\mathcal{A}} \equiv (\mathtt{y}\mathtt{Q}'\mathtt{V}'_1 ... \mathtt{V}'_m)^{\mathcal{A}}$, and $(\mathtt{y} :=\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}'_0\ \mathtt{else}\ \mathtt{N}'_1\ )$, is similar, but both rules (h) and ($\mathtt{if}$) must be used. □

Then we can state the soundness and completeness of the evaluation machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ with respect to the reduction on programs.

THEOREM 3.8. *Let* $\mathtt{M} \in \mathcal{P}$. *Then:*

(*1*) *If* $\models \mathtt{M} \Downarrow \mathtt{b}$ *then* $\mathtt{M} \to^*_{\beta\delta} \mathtt{b}$. (Soundness)
(*2*) *If* $\mathtt{M} \to^*_{\beta\delta} \mathtt{b}$ *then* $\models \mathtt{M} \Downarrow \mathtt{b}$. (Completeness)

PROOF.

(1) It follows directly by Lemma 3.7.(1).
(2) It follows directly by Lemma 3.7.(3). □

3.2. A small step version of $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$

In order to prove that programs are evaluated by the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ in polynomial space we need a formal definition of the space consumption, which in its turn needs a deep investigation on the machine behaviour. In fact, we will explicitly show that computations in the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ can be performed with no need of backtracking or complex state memorizations.

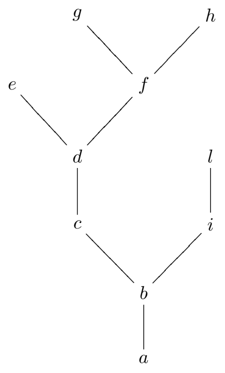
For this reason, in Table V.(a) we depict a small step abstract machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$. This new machine is able to reduce sequentially programs in $\mathrm{STA}_{\mathbf{B}}$ following a leftmost outermost strategy exploiting a contexts management similar to the one implemented by the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ but for the use of a garbage collector procedure, described in Table V.(b), in order to maintain the desired complexity property.

Definition 3.9 (*Small Step Machine* $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$).
The *reduction relation* $\mapsto \subseteq (\mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx}_{\mathrm{m}} \times \Lambda_{\mathcal{B}}) \times (\mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx}_{\mathrm{m}} \times \Lambda_{\mathcal{B}})$ is the relation inductively defined by the rules for the machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$ given in Table V. The relation $\mapsto^*$ is the reflexive and transitive closure of the reduction relation $\mapsto$.
If $\mathtt{M}$ is a program, and if there is a boolean $\mathtt{b}$ such that $\langle \circ, \varepsilon \succ \mathtt{M}\rangle \mapsto^* \langle \circ, \varepsilon \succ \mathtt{b}\rangle$ for some $\mathcal{A}$, then we say that $\mathtt{M}$ *reduces* to $\mathtt{b}$, and we simply write $\mathtt{M} \mapsto^* \mathtt{b}$ for short.

There is a direct correspondence between the configurations of a computation in the big step machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ and the ones in the small step machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$. This correspondence can be made explicit by the following translation:

Definition 3.10. Let $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$ be a configuration of a $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ computation. Then, the translation $(-)^{\mathsf{s}}$, assigning to ϕ a $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$ configuration, is defined as $(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{N}\rangle$ where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N})$.

The translation $(-)^{\mathsf{s}}$ defined above is useful in order to state the correspondence between the big step and the small step machine. In order to establish this correspondence we need to visit the evaluation trees of the big step machine computation following a determined visiting order. In particular, we consider the left-depth-first visit. E.g. consider the following tree:



the left-depth-first visit coincides with the visit of the nodes in the alphabetical order. Below, we need to talk about the visit of nodes in a given computation $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. For this reason, we say that a configuration ψ *immediately follows* a configuration ϕ if the node visited after ϕ for left-depth-first visit is the node ψ. For instance, the node i immediately follows the node h in the above figure.
Now we can state an important result.

LEMMA 3.11. *Let* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$ *and let* $\phi, \psi \in \nabla$ *be two distinct configurations (i.e.* $\phi \neq \psi$*) such that* ψ *immediately follows* ϕ *in the left-depth-first visit of* ∇*. Then:*

$$(\phi)^{\mathsf{s}} \mapsto (\psi)^{\mathsf{s}}$$

PROOF. We proceed by induction on the height of ∇. The base case is easy, since ∇ is an application of the (Ax) rule, hence there are no configurations $\phi, \psi \in \nabla$ such that $\phi \neq \psi$. Consider now the case where the height of ∇ is greater than 1. If the rule with conclusion ϕ is not an axiom, then ψ coincides with one of its premises. Let us consider all the possible cases. Consider the case where the rule with conclusion ϕ is (β). Then, we are in a situation as:

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

By definition of the translation $(-)^{\mathsf{s}}$ we have $(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ (\lambda\mathtt{x}.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, (\lambda\mathtt{x}.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)$ and $(\psi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}'' \succ \mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ where $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}, \mathcal{A}@[\mathtt{x}' := \mathtt{N}], \mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)$. By definition of the clear procedure it is also easy to verify that $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}, \mathcal{A}'@[\mathtt{x}' := \mathtt{N}], \mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)$. So, by one application of the rule (β_0) we have:

$$(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ (\lambda\mathtt{x}.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}, \mathcal{A}'' \succ \mathtt{P}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \rangle = (\psi)^{\mathsf{s}}$$

Consider the case where the rule with conclusion ϕ is:

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

Then, by definition of the translation $(-)^{\mathsf{s}}$ we have: $(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m)$, and similarly $(\psi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}'' \succ \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ where $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)$. Since $[\mathtt{x} := \mathtt{N}] \in \mathcal{A}$ and $\mathtt{x} \in \mathrm{FV}(\mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_n)$, by definition of the clear procedure it is easy to verify that $[\mathtt{x} := \mathtt{N}] \in \mathcal{A}'$ and thus $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}, \mathcal{A}', \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)$. So, by one application of the rule (h_0) we can conclude:

$$(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}, \mathcal{A}'' \succ \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle = (\psi)^{\mathsf{s}}$$

Consider the case where the rule with conclusion ϕ is:

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

Then, by definition we have $(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)$, and similarly $(\psi)^{\mathsf{s}} = \langle \mathcal{C}[\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A}'' \succ \mathtt{N} \rangle$ where $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}[\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A}, \mathtt{N})$. Obviously, we have $\mathcal{A}' = \mathcal{A}''$. Thus, by one application of the rule (if) we can conclude

$$\begin{aligned}(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto& \\ \langle \mathcal{C}[\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A}'' \succ \mathtt{N} \rangle = (\psi)^{\mathsf{s}}&\end{aligned}$$

The case where the rule with conclusion ϕ is (if 1) is similar.
Now consider the case ϕ is the conclusion of an axiom rule, i.e.:

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

If $\mathcal{C}$ is empty, then ϕ is the last configuration in the left-depth-first visit of ∇, hence there is no configuration $\psi \in \nabla$ immediately following ϕ such that $\phi \neq \psi$. Otherwise, ∇ has a subderivation $\diamond$ of the shape:

$$\frac{\dfrac{\dfrac{}{\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{b} \Downarrow \mathtt{b}}\ (Ax)}{\vdots} \qquad \psi \succ \mathcal{C}_1, \mathcal{A}_1 \models \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}_1}{\phi' \succ \mathcal{C}_1, \mathcal{A}_1 \models (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}_1}\ (\ \mathtt{if}\ \mathtt{b})$$

where by definition of left-depth-first visit ψ is the configuration immediately following ϕ. Note that $\mathtt{path}_{\diamond}(\phi)$ does not cross any if-rule by following its left premise. Hence, we have $\mathcal{C} = \mathcal{C}_1[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$.

By $(-)^{\mathsf{s}}$ definition we have $(\phi)^{\mathsf{s}} = \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{b}\rangle$ where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{b})$, and similarly $(\psi)^{\mathsf{s}} = \langle \mathcal{C}_1, \mathcal{A}'', \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n\rangle$ where $\mathcal{A}'' = \mathtt{clear}(\mathcal{C}_1, \mathcal{A}_1, \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n)$. Note that $\mathcal{A} = \mathcal{A}_1@[\mathtt{x}_1 := \mathtt{N}_1]@\cdots@[\mathtt{x}_k := \mathtt{N}_k]$ where the assignments $[\mathtt{x}_1 := \mathtt{N}_1], \dots, [\mathtt{x}_k := \mathtt{N}_k]$ have been introduced in the path from ϕ' to ϕ (by rules (β)). Since for each $1 \le i \le k$ we have $\mathtt{x}_i \notin \mathrm{FV}(\mathcal{C}) \cup \mathrm{FV}(\mathcal{A}_1) \cup \mathrm{FV}(\mathtt{b})$ it follows that $\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}_1, \mathtt{b})$. Thus, it is easy to verify that $\mathtt{clear}(\mathcal{C}_1, \mathcal{A}', \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n) = \mathtt{clear}(\mathcal{C}_1, \mathcal{A}_1, \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n) = \mathcal{A}''$. So, by one application of a rule $(r_{\mathtt{b}})$ we can conclude

$$(\phi)^{\mathsf{s}} = \langle \mathcal{C}_1[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A}' \succ \mathtt{b}\rangle \mapsto \langle \mathcal{C}_1, \mathcal{A}'', \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n\rangle = (\psi)^{\mathsf{s}}$$

and the proof is given. □

A converse of the above lemma can be easily obtained. Nevertheless, the previous result is sufficient in order to show that our space measures are sound. To this end we will use the following.

THEOREM 3.12. *Let* $\mathtt{M} \in \mathcal{P}$. *Then:*

$$\models \mathtt{M} \Downarrow \mathtt{b} \textit{ implies } \mathtt{M} \mapsto^* \mathtt{b}$$

PROOF. By using the translation $(-)^{\mathsf{s}}$ and by repeatedly applying Lemma 3.11. □

Indeed, Lemma 3.11, if repeatedly applied, allows us to define the execution in the $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ machine as a sequence of configurations corresponding to the left-depth-first visit of the derivation tree.

Example 3.13. By returning to the computation example in Table IV, it is worth noticing that to pass from the configuration ϕ to the configuration ψ all necessary information are already present in the configuration ϕ itself. That is, we have a machine $\mathrm{k}_{\mathcal{B}}^{\mathcal{C}}$ step as:

$$(\phi)^{\mathsf{s}} = \langle \mathcal{C}_0, \mathcal{A}' \succ \mathtt{0}\rangle \mapsto \langle [\circ], \mathcal{A}'' \succ \mathtt{x}_1\rangle = (\psi)^{\mathsf{s}}$$

where $\mathcal{A}' = \mathtt{clear}(\mathcal{C}_0, \mathcal{A}_3, 0) = \mathcal{A}_2$, and $\mathcal{A}'' = \mathtt{clear}([\circ], \mathcal{A}', \mathtt{x}_1) = \mathcal{A}_2$. So, more explicitly, since $\mathtt{clear}([\circ], \mathcal{A}_2, \mathtt{x}_1) = \mathcal{A}_2$, by applying a (r_0) rule we have

$$(\phi)^{\mathsf{s}} = \langle \mathtt{if}\ \circ\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1, \mathcal{A}_2 \succ \mathtt{0}\rangle \mapsto \langle [\circ], \mathcal{A}_2 \succ \mathtt{x}_1\rangle = (\psi)^{\mathsf{s}}$$

We can view such a step as a $\rightarrow_\delta$ reduction

$$(\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1\ )^{\mathcal{A}_3} = (\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1\ )^{\mathcal{A}_2} \rightarrow_\delta (\mathtt{x}_1)^{\mathcal{A}_2}$$

In fact, the behaviour shown in the above example is generalized by Lemma 3.11 and Theorem 3.12. So, in this sense we don't need neither a mechanism for backtracking nor the memorization of parts of the computation tree. Inspired by this property, we can define the notion of space needed to evaluate a term in the machines.

Let us first define the *size* of a configuration in both the machines.

Definition 3.14.

(1) If $\langle \mathcal{C}, \mathcal{A} \succ \mathtt{M}\rangle$ is a configuration in $\mathrm{k}_{\mathcal{B}}^{\mathcal{C}}$, then its *size* is $|\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}|$.
(2) If $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ is a configuration in $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$, then its *size* (denoted by $|\phi|$) is $|\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}|$.

We can now define the *required space* in both the machines as the maximal size of a configuration in the computation.

Definition 3.15.

(1) Let $\langle [\circ], \epsilon \succ \mathtt{M}\rangle \mapsto^* \mathtt{b}$ be a computation in $\mathrm{k}_{\mathcal{B}}^{\mathcal{C}}$. Then its *required space*, denoted by $\mathtt{space}_s(\mathtt{M})$, is the maximal size of a configuration in it.

(2) Let $\nabla :: [\circ], \epsilon \models \mathtt{M} \Downarrow \mathtt{b}$ be a computation in $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. Then its *required space*, denoted by $\mathtt{space}(\mathtt{M})$, is the maximal size of a configuration in ∇.

The small step machine is a concrete device to evaluate programs. Moreover, it is close to a concrete implementation. Indeed, the following is easy to prove.

LEMMA 3.16. *Each $\mathtt{M} \in \mathcal{P}$ can be evaluated using a Turing Machine working in space $O(\mathtt{space}_s(\mathtt{M})^2)$.*

PROOF. Easy. □

Note that in the previous lemma the square is a rough bound given to consider the encoding of the different data structures and the technique used to encode variables names, e.g. de Bruijn indexes. However, it is sufficient to prove polynomial space soundness in the next section.

We can now show that the relation on the required space of the two machines is the expected one.

LEMMA 3.17. *Let $\mathtt{M} \in \mathcal{P}$. Then:*

$$\mathtt{space}_s(\mathtt{M}) \leq \mathtt{space}(\mathtt{M})$$

PROOF. By definition of the translation $(-)^{\mathrm{s}}$ and Lemma 3.11. □

As a result, an upper bound on $\mathtt{space}(\mathtt{M})$ is also an upper bound on the amount of space which is needed to evaluate $\mathtt{M}$ using the small step machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$. So, from now on we can restrict our attention to prove the polynomial space measure soundness in the case of the big step evaluation machine.

4. PSPACE SOUNDNESS

In this section we will show that $\mathrm{STA}_{\mathbf{B}}$ is correct for polynomial space computations. The degree of a type derivation, i.e. the maximal nesting of applications of the rule (sp) in it, is the key notion in order to obtain the correctness. Indeed, we will prove that each program typable through a derivation with degree d can be executed on the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ in space polynomial in its size, where the maximum exponent of the polynomial is d. So, by considering fixed degrees we get PSPACE soundness. Considering a fixed d is not a limitation. In fact until now, in $\mathrm{STA}_{\mathbf{B}}$ programs we have not distinguished between the program code and input data. But it will be shown in Section 5 that data types are typable through derivations with degree 0. Hence, the degree can be considered as a real characteristic of the program code.
Moreover, every $\mathrm{STA}_{\mathbf{B}}$ program can be typed by means of several derivations with different degrees, nevertheless for each program there is a kind of minimal derivation for it, with respect to the degree. So, we can stratify programs with respect to the degree of their derivations, according to the following definition.

Definition 4.1.

(1) Let Π be a type derivation and Π_1 be one of its subderivations. The *depth* of Π_1 in Π, denoted $\mathtt{d}(\Pi_1, \Pi)$ is the number of applications of rule (sp) encountered in the path from the root of Π to the root of Π_1. It is inductively defined as follows:
— if Π coincides with the subderivation Π_1 then $\mathtt{d}(\Pi_1, \Pi) = 0$
— if Π ends by a rule

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

and Π_1 is a subderivation of Σ, then $\mathtt{d}(\Pi_1, \Pi) = \mathtt{d}(\Pi_1, \Sigma) + 1$

— in any other case Π ends by a rule (R) and Π_1 is a subderivation of one of the premise Σ of (R). In such a case, $\mathtt{d}(\Pi_1, \Pi) = \mathtt{d}(\Pi_1, \Sigma)$.

(2) Let Π be a type derivation. The *degree* of Π, denoted $\mathtt{d}(\Pi)$ is the maximal nesting of applications of rule (sp) in Π. That is,

$$\mathtt{d}(\Pi) = \max\{\mathtt{d}(\Sigma, \Pi) \mid \Sigma \text{ subderivation of } \Pi\}$$

(3) For each $d \in \mathbb{N}$ the set $\mathcal{P}_d$ is the set of $\mathrm{STA}_{\mathbf{B}}$ programs typable through derivation with degree d.

$$\mathcal{P}_d = \{\mathtt{M} \mid \ \Pi \rhd \vdash \mathtt{M} : \mathbf{B} \ \wedge \ \mathtt{d}(\Pi) = d\}$$

Clearly, $\mathcal{P}$ corresponds to the union for $n \in \mathbb{N}$ of the different $\mathcal{P}_n$.

The proof of the PSPACE soundness is quite involved, for this reason we divide this section into three subsections. In the first, we refine the connections between the *required space* of a computation as defined in the previous section and the behaviour of the $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ machine. In the second one, we define a weight associated to machine configurations. This notion of weight is obtained by extending some measures defined on type derivations to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. Finally, in the third subsection, the soundness with respect to PSPACE will be proved. The key ingredient will be Lemma 4.12, establishing that the configuration weights decrease in exploring a computation path.

4.1. Refining $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ Space Measures

In this subsection we establish some relations between the size of the contexts in a configuration and the behaviour of the machine. These relations will then be useful in the next subsections to prove the PSPACE soundness.

Definition 4.2. Let ∇ be a computation and $\phi \in \nabla$ a configuration. Then:

— the symbol $\#_{\beta}(\phi)$ denotes the number of applications of the (β) rule in $\mathtt{path}(\phi)$,
— the symbol $\#_{h}(\phi)$ denotes the number of applications of the (h) rule in $\mathtt{path}(\phi)$,
— the symbol $\#_{\mathtt{if}}(\phi)$ denotes the number of applications of $(\mathtt{if}\ 0)$ and $(\mathtt{if}\ 1)$ rules in $\mathtt{path}(\phi)$.

The cardinality of the contexts in a configuration ϕ is a measure of the number of some rules performed by the machine in the path to reach ϕ.

LEMMA 4.3. *Let* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$ *be a computation. Then, for each configuration* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$*:*

(*1*) $\#(\mathcal{A}) = \#_{\beta}(\phi)$
(*2*) $\#(\mathcal{C}) = \#_{\mathtt{if}}(\phi)$

PROOF.

(1) Easy, by induction on the length of $\mathtt{path}(\phi)$, since m-contexts can grow only through applications of the (β) rule.
(2) Easy, by induction on the length of $\mathtt{path}(\phi)$, since $\mathbf{B}$-contexts can grow only through applications of $(\ \mathtt{if}\ 0)$ and $(\ \mathtt{if}\ 1)$ rules. $\Box$

The following key property ensures that in each computation ∇ of the shape $\models \mathtt{M} \Downarrow \mathtt{b}$ only subterms of the initial term $\mathtt{M}$ are recorded in the m-contexts.

PROPERTY 4.4. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$ *if* $[\mathtt{x} := \mathtt{N}] \in \mathcal{A}$ *then* $\mathtt{N}$ *is an instance (possibly with renaming of variables) of a subterm of* $\mathtt{M}$.

PROOF. The property is proved by contradiction. Take the configuration ψ with minimal path from it to the root of ∇, such that in its m-context $\mathcal{A}_\psi$ there is $\mathtt{x} := \mathtt{N}$, where $\mathtt{N}$ is not an instance of a subterm of $\mathtt{M}$. Let p be the length of this path. Since the only rule that makes the m-context grow is a (β) rule we are in a situation like the following:

$$\frac{\psi \succ \mathcal{C}, \mathcal{A}'@[\mathtt{x} := \mathtt{N}] \models \mathtt{P}[\mathtt{x}/\mathtt{x}']\mathtt{V}_1 \cdots \mathtt{V}_n \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A}' \models (\lambda \mathtt{x}'.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_n \Downarrow \mathtt{b}}$$

If $\mathtt{N}$ is not an instance of a subterm of $\mathtt{M}$ it has been obtained by a substitution. Substitutions can be made only through applications of rule (h) replacing the head variable. Hence, by the shape of $(\lambda \mathtt{x}'.\mathtt{P})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_n$, the only possible situation is that there exists an application of rule (h) as:

$$\frac{[\mathtt{y} := \mathtt{M}'] \in \mathcal{A}' \quad \mathcal{C}, \mathcal{A}' \models \mathtt{M}'\mathtt{V}'_1 \cdots \mathtt{V}'_m \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A}' \models \mathtt{y}\mathtt{V}'_1 \cdots \mathtt{V}'_m \Downarrow \mathtt{b}}$$

with $\mathtt{N}$ a subterm of $\mathtt{M}'$. But this implies $\mathtt{M}'$ is not an instance of a subterm of $\mathtt{M}$ and it has been introduced by a rule of a path of length less than p, contradicting the hypothesis. $\square$

The next lemma gives upper bounds on the size of the m-context, of the $\mathbf{B}$-context and of the subject of a configuration.

LEMMA 4.5. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$ *then for each configuration* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$*:*

(*1*) $|\mathcal{A}| \leq \#_\beta(\phi) \times (|\mathtt{M}| + 1)$
(*2*) $|\mathtt{P}| \leq (\#_h(\phi) + 1) \times |\mathtt{M}|$
(*3*) $|\mathcal{C}| \leq \#_{\mathtt{if}}(\phi) \times (\max\{|\mathtt{N}| \mid \psi \succ \mathcal{C}', \mathcal{A}' \models \mathtt{N} \Downarrow \mathtt{b}'' \in \mathtt{path}(\phi)\})$

PROOF.

(1) By inspection of the rules of Table III it is easy to verify that m-contexts can grow only by applications of the (β) rule. So the conclusion follows by Lemma 4.3.1 and Property 4.4.
(2) By inspection of the rules of Table III it is easy to verify that the subject can grow only by substitutions through applications of the (h) rule. So the conclusion follows by Property 4.4.
(3) By inspection of the rules of Table III it is easy to verify that $\mathbf{B}$-contexts can grow only by applications of $(\mathtt{if}\ 0)$ and $(\mathtt{if}\ 1)$ rules. So the conclusion follows directly by Lemma 4.3.2. $\square$

4.2. Space Weight and $\mathrm{STA}_\mathbf{B}$

In order to prove the PSPACE soundness in the next subsection we need to establish a formal connection between the machine configurations and the information given by the type system. To some extent this connection has been already established by Lemma 3.7. We here make it more explicit thanks to the following property.

PROPERTY 4.6. *Let* $\mathtt{P} \in \mathcal{P}$ *such that* $\Pi \rhd \vdash \mathtt{P} : \mathbf{B}$ *and* $\nabla :: \models \mathtt{P} \Downarrow \mathtt{b}$. *Then, for each configuration* $\phi \succ \mathcal{C}, [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n : \mathtt{N}_n] \models \mathtt{N} \Downarrow \mathtt{b}_1 \in \nabla$*:*

(*1*) *there is a type derivation* $\Theta \rhd \mathtt{x}_1 :!^{m_1} A_1, \ldots, \mathtt{x}_n :!^{m_n} A_n \vdash \mathtt{N} : \mathbf{B}$ *obtained by composing and duplicating some subderivations of* Π *(possibly with renaming of variables).*
(*2*) *there are type derivations* $\Sigma_1, \ldots, \Sigma_n$ *such that for each* $1 \leq i \leq n$*:* $\Sigma_i \rhd \Gamma_i \vdash \mathtt{N}_i : A_i$ *and* Σ_i *is a subderivation of* Π *(possibly with renaming of variables).*

(3) *there are type derivations* $\Theta_i \rhd \mathtt{x}_i :!^{m_i} A_i, \ldots, \mathtt{x}_n :!^{m_n} A_n \vdash \mathtt{N}^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$ *for* $1 \leq i \leq n$ *obtained by composing and duplicating some subderivations of* Π *(possibly with renaming of variables).*

PROOF. All the three points follows easily by Lemma 3.7.2, using also Lemma 2.6.3 in the case of Point (2). Moreover, note that in Point (1) we can assume that all the variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$ appear in Θ thanks to the (w) rule. The same is true in Point (3) for the variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$. □

The definition of space weight for a configuration is slightly involved. We first start by adapting to our case some measures for Soft Linear Logic given by Lafont in [Lafont 2004].

Definition 4.7.

— The *rank* of a rule (m):

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

is the number $k \leq n$ of variables $\mathtt{x}_i$ such that $\mathtt{x}_i$ belongs to the free variables of $\mathtt{M}$. Let r be the the maximal rank of a rule (m) in Π. Then, the rank of Π is $\mathtt{rk}(\Pi) = \max(r, 1)$.

— Let r be a natural number. The *slice weight* $\delta(\Pi, r)$ of Π with respect to r is defined inductively as follows:

— If Π consists in an (Ax) or in a $(\mathbf{B}_\mathrm{b} I)$ rule, then $\delta(\Pi, r) = 1$.

— If Π ends by a rule

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

then $\delta(\Pi, r) = \delta(\Sigma, r) + 1$.

— If Π ends by a rule

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

then $\delta(\Pi, r) = r \ \times\ \delta(\Sigma, r)$.

— If Π ends by a rule

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

then $\delta(\Pi, r) = \delta(\Sigma, r) + \delta(\Theta, r) + 1$.

— If Π ends by a rule

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

then $\delta(\Pi, r) = \max\{\delta(\Sigma, r), \delta(\Theta_0, r), \delta(\Theta_1, r)\} + 1$

— In any other case $\delta(\Pi, r) = \delta(\Sigma, r)$ where Σ is the unique premise derivation.

The sliced weight of a type derivation is useful to control the space that can be used during a computation. However, it does not suffice. We also need a measure expressing the *increasing of the slice weight* due to a substitution. This is the *factor* of a variable in a type derivation.

Definition 4.8. Let $\Pi \rhd \Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau$, then the *factor* of $\mathtt{x}$ in Π with respect to r, denoted $\gamma(\mathtt{x}, \Pi, r)$ is inductively defined as follows:

— If Π consists in an (Ax) rule introducing $\mathtt{x}$ as

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

then $\gamma(\mathtt{x}, \Pi, r) = 1$.

— If Π consists in an (Ax) rule not introducing $\mathtt{x}$, or in a (w) rule introducing $\mathtt{x}$, or in a $(\mathbf{B}_{\mathtt{b}}I)$ rule, then $\gamma(\mathtt{x}, \Pi, r) = 0$.

— If Π ends by a rule

$$\frac{\Sigma \rhd \Gamma, \mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_1 \vdash \mathtt{N} : \tau}{\Gamma, x :!\sigma_1 \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \tau}\ (m)$$

then $\gamma(\mathtt{x}, \Pi, r) = \sum_{1 \leq i \leq n} \gamma(\mathtt{x}_i, \Sigma, r)$.

— If Π ends by a rule

$$\frac{\Sigma \rhd \Gamma, \mathtt{x} : \sigma_1 \vdash \mathtt{N} : \tau}{!\Gamma, \mathtt{x} :!\sigma_1 \vdash \mathtt{N} :!\tau}\ (sp)$$

then $\gamma(\mathtt{x}, \Pi, r) = r \times \gamma(\mathtt{x}, \Sigma, r)$.

— If Π ends by a rule

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

then $\gamma(\mathtt{x}, \Pi, r) = \gamma(\mathtt{x}, \Sigma, r)$ if $\mathtt{x} \in \mathrm{dom}(\Gamma)$, $\gamma(\mathtt{x}, \Pi, r) = \gamma(\mathtt{x}, \Theta, r)$ otherwise.

— If Π ends by a rule

$$\frac{\Sigma_1 \rhd \Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : \mathbf{B} \quad \Sigma_2 \rhd \Gamma, \mathtt{x} : \sigma \vdash \mathtt{N}_0 : A \quad \Sigma_3 \rhd \Gamma, \mathtt{x} : \sigma \vdash \mathtt{N}_1 : A}{\Gamma, \mathtt{x} : \sigma \vdash \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A}$$

then $\gamma(\mathtt{x}, \Pi, r) = \max\{\gamma(\mathtt{x}, \Sigma_1, r), \gamma(\mathtt{x}, \Sigma_2, r), \gamma(\mathtt{x}, \Sigma_3, r)\}$

— In any other case $\gamma(\mathtt{x}, \Pi, r) = \gamma(\mathtt{x}, \Sigma, r)$ where Σ is the unique premise derivation.

The previous definition can also be understood as a generalization of the concept of *sliced occurrences* of a variable used instead in [Gaboardi et al. 2008a].

Now we have all the ingredients to define the needed measure for configurations, which is represented by the *space weight* defined as follows.

Definition 4.9 (*Space Weight*). Let $\mathtt{P} \in \mathcal{P}$ such that $\Pi \rhd \vdash \mathtt{P} : \mathbf{B}$ and $\nabla :: \models \mathtt{P} \Downarrow \mathtt{b}$. Then, the *space weight of a configuration* $\phi \succ \mathcal{C}, [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n : \mathtt{N}_n] \models \mathtt{N} \Downarrow \mathtt{b}_1 \in \nabla$, denoted $\delta_\Pi(\phi, r)$ is defined as:

$$\delta_\Pi(\phi, r) = \delta(\Sigma, r) + \gamma(\mathtt{x}_1, \Theta_1, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n, r) \times \delta(\Sigma_n, r)$$

where $\Sigma \rhd \mathtt{x}_1 :!^{m_1} A_1, \ldots, \mathtt{x}_n :!^{m_n} A_n \vdash \mathtt{N} : \mathbf{B}$, $\Sigma_i \rhd \Gamma_i \vdash \mathtt{N}_i : A_i$ and $\Theta_i \rhd \mathtt{x}_i :!^{m_i} A_i, \ldots, \mathtt{x}_n : !^{m_n} A_n \vdash \mathtt{N}^{[\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_{i-1} := \mathtt{N}_{i-1}]} : \mathbf{B}$ for $1 \leq i \leq n$.

The definition above could seem a bit odd. First, note that it is well defined thanks to Property 4.6. Besides, one can read it as the sum of the weights given by the different terms in a configuration where the weight of each term $\mathtt{N}_i$ in the m-context is multiplied by the actual factor of the corresponding variable $\mathtt{x}_i$. Clearly, we have the following.

LEMMA 4.10. *If* $\Pi \rhd \vdash \mathtt{P} : \mathbf{B}$ *and* $\nabla :: \models \mathtt{P} \Downarrow \mathtt{b}$, *then for the initial configuration* $\phi \succ \models \mathtt{P} \Downarrow \mathtt{b}$ *we have*

$$\delta_\Pi(\phi, r) = \delta(\Pi, r)$$

PROOF. Easy. □

Moreover, another important property of the measures we have defined above is the following.

LEMMA 4.11. *Let* $\Pi \rhd \Gamma, \mathtt{x} :!^n A \vdash \mathtt{M} : \tau$*, then for each* $r \geq \mathtt{rk}(\Pi)$*:*

$$\gamma(\mathtt{x}, \Pi, r) \leq r^n$$

PROOF. It follows easily by induction on Π and definition of rank. □

4.3. Proof of PSPACE Soundness

We are now ready to show that the space weight δ gives a bound on the number of rules in a computation path of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$.

LEMMA 4.12. *Let* $\mathtt{P} \in \mathcal{P}$ *with* $\Pi \rhd \vdash \mathtt{P} : \mathbf{B}$ *and* $\nabla :: \models \mathtt{P} \Downarrow \mathtt{b}$.

(*1*) *Consider an occurrence in* ∇ *of the rule:*

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

Then, for every $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta_\Pi(\phi, r) > \delta_\Pi(\psi, r)$$

(*2*) *Consider an occurrence in* ∇ *of an* `if` *rule as:*

$$\frac{\psi_1 \succ \mathcal{C}', \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b} \quad \psi_2 \succ \mathcal{C}, \mathcal{A} \models \mathtt{N}_\mathtt{b}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}{\phi \succ \mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'} \ (\ \mathtt{if}\ \mathtt{b})$$

where $\mathcal{C}' \equiv \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$*. Then, for every* $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta_\Pi(\phi, r) > \delta_\Pi(\psi_1, r) \quad \textit{and} \quad \delta_\Pi(\phi, r) > \delta_\Pi(\psi_2, r)$$

(*3*) *Consider an occurrence in* ∇ *of an* h *rule as:*

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

Then, for every $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta_\Pi(\phi, r) > \delta_\Pi(\psi, r)$$

PROOF.

(1) We proceed by induction on m. Consider the case $m = 0$ and suppose $\Sigma \rhd \Gamma \vdash (\lambda \mathtt{x}.\mathtt{M})\mathtt{N} : \mathbf{B}$ and $\Theta \rhd \Gamma, \mathtt{x}' : \tau \vdash \mathtt{M}[\mathtt{x}'/\mathtt{x}] : \mathbf{B}$. Moreover, assume $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n := \mathtt{N}_n]$ with $\Sigma_i \rhd \Delta_i \vdash \mathtt{N}_i : A_i$ for $1 \leq i \leq n$. Without loss of generality we can assume that Σ ends as follows:

$$\dfrac{\dfrac{\Pi_1 \rhd \Gamma_1, \mathtt{x} :!^k A \vdash \mathtt{M} : \mathbf{B}}{\Gamma_1 \vdash \lambda \mathtt{x}.\mathtt{M} :!^k A \multimap \mathbf{B}} \ (\multimap I) \quad \dfrac{\Pi_2 \rhd \Gamma_2 \vdash \mathtt{N} : A}{!^k\Gamma_2 \vdash \mathtt{N} :!^k A} \ (sp)}{\Gamma_1, !^k\Gamma_2 \vdash (\lambda \mathtt{x}.\mathtt{M})\mathtt{N} : \mathbf{B}} \ (\multimap E)$$

where $\Gamma_1 \# !^k\Gamma_2$. Since we have $\delta(\Sigma, r) = \delta(\Pi_1, r) + r^k\delta(\Pi_2, r) + 2$, by definition of space weight we have:

$$\delta_\Pi(\phi, r) = \delta(\Pi_1, r) + r^k \times \delta(\Pi_2, r) + \gamma(\mathtt{x}_1, \Theta^1_1, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta^1_n, r) \times \delta(\Sigma_n, r) + 2$$

where $\Theta^1_i \rhd \Delta^1_i \vdash ((\lambda \mathtt{x}.\mathtt{M})\mathtt{N})^{[\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_{i-1} := \mathtt{N}_{i-1}]} : \mathbf{B}$ for $1 \leq i \leq n$. Analogously, since clearly $\delta(\Theta, r) = \delta(\Pi_1, r)$ we have:

$$\delta_\Pi(\psi, r) = \delta(\Pi_1, r) + \gamma(\mathtt{x}', \Pi_1, r) \times \delta(\Pi_2, r) + \gamma(\mathtt{x}_1, \Theta^2_1, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta^2_n, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^2 \rhd \Delta_i^2 \vdash (\mathtt{M}[\mathtt{x}'/\mathtt{x}])^{[\mathtt{x}':=\mathtt{N},\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$, for $1 \leq i \leq n$.
Since the occurrences of the rules introducing $\mathtt{x}'$ can only be at a depth less than k, it is easy to verify that for each $1 \leq i \leq n$:

$$\gamma(\mathtt{x}_i, \Theta_i^2, r) \leq \gamma(\mathtt{x}_i, \Theta_i^1, r)$$

Moreover, since $r \geq \mathtt{rk}(\Pi)$ by Lemma 4.11 we have $\gamma(\mathtt{x}', \Pi_1, r) \leq r^k$, and so we can conclude

$$\delta_\Pi(\psi, r) \leq \delta_\Pi(\phi, r) - 2$$

The inductive step $m = k + 1$ follows easily by the induction hypothesis.

(2) Suppose $\Sigma \rhd \Delta \vdash (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m : \mathbf{B}$, $\Sigma^1 \rhd \Delta \vdash \mathtt{M} : \mathbf{B}$ and $\Sigma^2 \rhd \Delta \vdash \mathtt{N}_\mathtt{b}\mathtt{V}_1 \cdots \mathtt{V}_m : \mathbf{B}$. Moreover, assume $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n := \mathtt{N}_n]$ and that $\Sigma_i \rhd \Gamma_i \vdash \mathtt{N}_i : A_i$ for $1 \leq i \leq n$. By definition of space weight we have:

$$\delta_\Pi(\phi, r) = \delta(\Sigma, r) + \gamma(\mathtt{x}_1, \Theta_1^1, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^1, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^1 \rhd \Delta_i^1 \vdash ((\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$ for $1 \leq i \leq n$. Analogously, we have:

$$\delta_\Pi(\psi_1, r) = \delta(\Sigma^1, n) + \gamma(\mathtt{x}_1, \Theta_1^2, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^2, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^2 \rhd \Delta_i^2 \vdash (\mathtt{M})^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$, for $1 \leq i \leq n$. Finally we also have:

$$\delta_\Pi(\psi_2, r) = \delta(\Sigma^2, n) + \gamma(\mathtt{x}_1, \Theta_1^3, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^3, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^3 \rhd \Delta_i^3 \vdash (\mathtt{N}_\mathtt{b}\mathtt{V}_1 \cdots \mathtt{V}_m)^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$, for $1 \leq i \leq n$.
By definition of the factor of a variable, it is easy to verify that for $1 \leq i \leq n$ we have both

$$\gamma(\mathtt{x}_i, \Theta_i^1, r) \geq \gamma(\mathtt{x}_i, \Theta_i^2, r) \quad \text{and} \quad \gamma(\mathtt{x}_i, \Theta_i^1, r) \geq \gamma(\mathtt{x}_i, \Theta_i^3, r)$$

Moreover, by definition of slice weight we have

$$\delta(\Sigma, r) > \delta(\Sigma^1, r) \quad \text{and} \quad \delta(\Sigma, r) > \delta(\Sigma^2, r)$$

So, we can conclude both

$$\delta_\Pi(\phi, r) > \delta_\Pi(\psi_1, r) \quad \text{and} \quad \delta_\Pi(\phi, r) > \delta_\Pi(\psi_2, r)$$

(3) Suppose $\Sigma \rhd \Delta \vdash \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m : \mathbf{B}$ and $\Theta \rhd \Delta \vdash \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m : \mathbf{B}$. Moreover, assume $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n := \mathtt{N}_n]$ with $\mathtt{x} = \mathtt{x}_k$ and $\mathtt{N} = \mathtt{N}_k$ for some $1 \leq k \leq n$ and that $\Sigma_i \rhd \Gamma_i \vdash \mathtt{N}_i : A_i$. By definition of space weight we have:

$$\delta_\Pi(\phi, r) = \delta(\Sigma, r) + \gamma(\mathtt{x}_1, \Theta_1^1, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^1, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^1 \rhd \Delta_i^1 \vdash (\mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m)^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$ for $1 \leq i \leq n$. Analogously, we have:

$$\delta_\Pi(\psi, r) = \delta(\Theta, n) + \gamma(\mathtt{x}_1, \Theta_1^2, r) \times \delta(\Sigma_1, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^2, r) \times \delta(\Sigma_n, r)$$

where $\Theta_i^2 \rhd \Delta_i^2 \vdash (\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{[\mathtt{x}_1:=\mathtt{N}_1,\ldots,\mathtt{x}_{i-1}:=\mathtt{N}_{i-1}]} : \mathbf{B}$, for $1 \leq i \leq n$.
Let d be the depth in Σ of the axiom introducing the occurrence of $\mathtt{x}$ that is replaced by $\mathtt{N}$ in Θ. Noting that such an axiom cannot be above a $(\mathbf{B}E)$ rule in Σ, we have

$$\delta(\Theta, r) = \delta(\Sigma, r) - r^d + r^d \times \delta(\Sigma_k, r)$$

Now, it is easy to verify that $\gamma(\mathtt{x}_i, \Theta_i^1, r) = \gamma(\mathtt{x}_i, \Theta_i^2, r)$ for all $1 \leq i \leq n$ with $i \neq k$ thanks to the fact that $\mathtt{N}$ can contain only $\mathtt{x}_j$ for $j > k$. However, we have $\gamma(\mathtt{x}_k, \Theta_k^2, r) = \gamma(\mathtt{x}_i, \Theta_k^1, r) - r^d$ So, summarizing we have

$$\begin{aligned}\delta_\Pi(\psi, r) = (\delta(\Sigma, r) - r^d + r^d \times \delta(\Sigma_k, r)) + \gamma(\mathtt{x}_1, \Theta_1^1, r) \times \delta(\Sigma_1, r) + \cdots +\\ (\gamma(\mathtt{x}_k, \Theta_k^1, r) - r^d) \times \delta(\Sigma_k, r) + \cdots + \gamma(\mathtt{x}_n, \Theta_n^1, r) \times \delta(\Sigma_n, r)\end{aligned}$$

and since clearly $r^d \geq 1$ we can conclude

$$\delta_\Pi(\psi, r) \leq \delta_\Pi(\phi, r) - 1$$

□

Thanks to the above lemma we have the following important property.

LEMMA 4.13. *Let* $\Pi \rhd \mathtt{M} : \mathbf{B}$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \in \nabla$ *such that* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$ *if* $r \geq \mathtt{rk}(\Pi)$*:*

$$\#_\beta(\phi) + \#_{\mathtt{if}}(\phi) + \#_h(\phi) \leq \delta(\Pi, r)$$

PROOF. Easy, by Lemma 4.12 and Lemma 4.10. □

As defined in the previous section, the space used by the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ is the maximum space used by its configurations. The above lemma gives a measure on the number of rules in the path to reach a configuration. We now only need to relate the space weight with both the size of the term and the degree of the derivation. This is done thanks to the following lemma.

LEMMA 4.14. *Let* $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$.

(*1*) $\delta(\Pi, 1) \leq |\mathtt{M}|$.
(*2*) $\delta(\Pi, r) \leq \delta(\Pi, 1) \times r^{\mathtt{d}(\Pi)}$.
(*3*) $\delta(\Pi, \mathtt{rk}(\Pi)) \leq |\mathtt{M}|^{\mathtt{d}(\Pi)+1}$.

PROOF.

(1) By induction on Π. Base cases are trivial. Cases $(sp), (m), (w), (\forall I)$ and $(\forall E)$ follow directly by induction hypothesis. The other cases follow by definition of $\delta(\Pi, 1)$.
(2) By induction on Π. Base cases are trivial. Cases $(m), (w), (\forall I)$ and $(\forall E)$ follow directly by induction hypothesis. The other cases follow by induction hypothesis and the definitions of $\delta(\Pi, r)$ and $\mathtt{d}(\Pi)$.
(3) By definition of rank it is easy to verify that $\mathtt{rk}(\Pi) \leq |\mathtt{M}|$, hence by the previous two points the conclusion follows. □

The next lemma gives a bound on the dimension of all the components of a machine configuration, namely the term, the m-context and the $\mathbf{B}$-context.

LEMMA 4.15. *Let* $\mathtt{M} \in \mathcal{P}_d$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$*:*

$$|\mathcal{A}| + |\mathtt{N}| + |\mathcal{C}| \leq 2 \times |\mathtt{M}|^{2d+3}$$

PROOF. By Lemma 4.5 we have

$$\begin{aligned} |\mathcal{A}| + |\mathtt{N}| + |\mathcal{C}| \leq\ & \#_\beta(\phi) \times (|\mathtt{M}| + 1) + (\#_h(\phi) + 1) \times |\mathtt{M}| + \\ & \#_{\mathtt{if}}(\phi) \times (\max\{|\mathtt{N}| \mid \psi \succ \mathcal{C}', \mathcal{A}' \models \mathtt{N} \Downarrow \mathtt{b}'' \in \mathtt{path}(\phi)\}) \\ \leq\ & (\#_\beta(\phi) + (\#_h(\phi) + 1) + \#_{\mathtt{if}}(\phi)) \times ((\#_h(\phi) + 1) \times |\mathtt{M}|) \end{aligned}$$

So, by Lemma 4.13 and Lemma 4.14 we can conclude.

$$\begin{aligned} |\mathcal{A}| + |\mathtt{N}| + |\mathcal{C}| &\leq \delta(\Pi, \mathtt{rk}(\Pi)) \times ((\delta(\Pi, \mathtt{rk}(\Pi)) + 1) \times |\mathtt{M}|) \\ &\leq |\mathtt{M}|^{\mathtt{d}+1} \times ((|\mathtt{M}|^{\mathtt{d}+1} + 1) \times |\mathtt{M}|) \leq 2 \times |\mathtt{M}|^{2\mathtt{d}+3} \quad \square \end{aligned}$$

The above lemma gives us the bound on the space required to evaluate a program by means of the $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ machine.

THEOREM 4.16. *Let* $\mathtt{M} \in \mathcal{P}_d$. *Then:*

$$\mathtt{space}(\mathtt{M}) \leq 2 \times |\mathtt{M}|^{2d+3}$$

PROOF. By definition of $\mathtt{space}(\mathtt{M})$ and Lemma 4.15. □

Finally the PSPACE soundness follows immediately.

THEOREM 4.17 (POLYNOMIAL SPACE SOUNDNESS).
Each $\mathtt{M} \in \mathcal{P}_d$ *can be evaluated by a Turing Machine working in space* $O(|\mathtt{M}|^{4d+6})$.

PROOF. By Theorem 4.16, Lemma 3.17 and Lemma 3.16. □

5. PSPACE COMPLETENESS

A well known result of the seventies states that the class of problem decidable by a Deterministic Turing Machine (DTM) in space polynomial in the length of the input coincides with the class of problems decidable by an Alternating Turing Machine (ATM) [Chandra et al. 1981] in time polynomial in the length of the input, i.e.

$$\text{PSPACE} = \text{APTIME}$$

We use this result, and we prove that each polynomial time ATM $\mathcal{M}$ can be simulated by a term typable in $\text{STA}_{\mathbf{B}}$. In order to do this, we will use a result already obtained by two of the authors of this paper [Gaboardi and Ronchi Della Rocca 2007; Gaboardi 2007], namely that STA, the type assignment system for the λ-calculus on which $\text{STA}_{\mathbf{B}}$ is based, characterizes all the polynomial time functions. In particular, we use the same encoding as in [Gaboardi and Ronchi Della Rocca 2007; Gaboardi 2007] for the representation of the polynomials. Notice that the data types are coded by means of terms that are typable in a uniform way through derivations of degree 0. This approach ensures that the degree of the polynomial space bound does not depends on the input data.

Some syntactic sugar

Let $\circ$ denotes composition. In particular $\mathtt{M} \circ \mathtt{N}$ stands for $\lambda\mathtt{z}.\mathtt{M}(\mathtt{N}\mathtt{z})$ 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 $\lambda\mathtt{x}.\mathtt{x}\mathtt{M}\mathtt{N}$ and $\mathbf{let}\ \mathtt{z}\ \mathbf{be}\ \mathtt{x}, \mathtt{y}\ \mathbf{in}\ \mathtt{N}$ stands for $\mathtt{z}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{N})$. Note that, since $\text{STA}_{\mathbf{B}}$ is an affine system, tensor product enjoys some properties of the additive conjunction, as to permit the projections: as usual $\pi_1(\mathtt{M})$ stands for $\mathtt{M}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x})$ and $\pi_2(\mathtt{M})$ stands for $\mathtt{M}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y})$. The 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. In the sequel we sometimes consider tensor product modulo associativity.

B-programmable functions

We need both to generalize the usual notion of lambda definability, given in [Barendregt 1984], to different kinds of input data, and to specialize it to our typing system.

Definition 5.1. Let $f : \mathbb{I}_1 \times \cdots \times \mathbb{I}_n \to \mathbb{O}$ be a total function and let each element $o \in \mathbb{O}$ and $i_j \in \mathbb{I}_j$, for $0 \leq j \leq n$, be encoded by terms $\underline{\mathtt{o}}$ and $\underline{\mathtt{i}_j}$ such that $\vdash \underline{\mathtt{o}} : \mathbf{O}$ and $\vdash \underline{\mathtt{i}_j} : \mathbf{I}_j$.

(i) The function f is $\mathbf{B}$-*definable* if there is a term $\underline{\mathtt{f}} \in \Lambda_{\mathcal{B}}$ such that for each $i_j \in \mathbb{I}_j$ the term $\underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n}$ is typable as $\vdash \underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n} : \mathbf{O}$ and:

$$f(i_1, \ldots, i_n) = o \iff \underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n} =_{\beta\delta} \underline{\mathtt{o}}$$

(ii) Let $\mathbb{O} = \mathbf{B}$. The function f is $\mathbf{B}$-programmable if there is a term $\underline{\mathtt{f}} \in \Lambda_{\mathcal{B}}$ such that for each $i_j \in \mathbb{I}_j$: $\underline{\mathtt{f}}\underline{\mathtt{i}_1} \ldots \underline{\mathtt{i}_n} \in \mathcal{P}$ and:

$$f(i_1, \ldots, i_n) = b \iff \models \underline{\mathtt{f}}\underline{\mathtt{i}_1} \ldots \underline{\mathtt{i}_n} \Downarrow \underline{\mathtt{b}}$$

Natural numbers and strings of booleans

Natural numbers, as usual in the λ-calculus, are represented by Church numerals, i.e. $\underline{\mathtt{n}} \doteq \lambda\mathtt{s}.\lambda\mathtt{z}.\mathtt{s}^n(\mathtt{z})$. Each Church numeral $\underline{\mathtt{n}}$ is such that $\vdash \underline{\mathtt{n}} : \mathbf{N}_i$ for every $i \geq 1$ where the *indexed type* $\mathbf{N}_i$ is defined as:

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

It is easy to check that $\underline{\mathtt{n}}$ is typable by means of derivations with degree 0. We simply use $\mathbf{N}$ to mean $\mathbf{N}_1$.
The standard terms $\mathtt{suc} \doteq \lambda\mathtt{n}.\lambda\mathtt{s}.\lambda\mathtt{z}.\mathtt{s}(\mathtt{nsz})$, $\mathtt{add} \doteq \lambda\mathtt{n}.\lambda\mathtt{m}.\lambda\mathtt{s}.\lambda\mathtt{z}.\mathtt{ns}(\mathtt{msz})$ and $\mathtt{mul} \doteq \lambda\mathtt{n}.\lambda\mathtt{m}.\lambda\mathtt{s}.\mathtt{n}(\mathtt{ms})$, defining successor, addition and multiplication, analogously to what happens in STA, are typable as: $\vdash \mathtt{suc} : \mathbf{N}_i \multimap \mathbf{N}_{i+1}$, $\vdash \mathtt{add} : \mathbf{N}_i \multimap \mathbf{N}_j \multimap \mathbf{N}_{\max(i,j)+1}$ and $\vdash \mathtt{mul} : \mathbf{N}_i \multimap !^i\mathbf{N}_j \multimap \mathbf{N}_{i+j}$. From this we have for $\mathrm{STA}_{\mathbf{B}}$ the following completeness for polynomials.

LEMMA 5.2 ([GABOARDI AND RONCHI DELLA ROCCA 2007]). *Let P be a polynomial and $deg(P)$ its degree. Then there is a term* $\mathtt{P}$ *defining P typable as:*

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

Strings of booleans are represented by terms of the shape $\lambda\mathtt{c}.\lambda\mathtt{z}.\mathtt{cb}_0(\cdots(\mathtt{cb}_n\mathtt{z})\cdots)$ 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 $\mathtt{len} \doteq \lambda\mathtt{c}.\lambda\mathtt{s}.\mathtt{c}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{sy})$ typable as $\vdash \mathtt{len} : \mathbf{S}_i \multimap \mathbf{N}_i$ that given a string of booleans returns its length. Note that the data types defined above can be typed in $\mathrm{STA}_{\mathbf{B}}$ by derivations with degree 0.

Boolean connectives

Thanks to the presence of the $(\mathbf{B}E)$ rule it is possible to define the usual boolean connectives. Remembering that in our language `0` denotes “true” while `1` denotes “false”, we have the following terms:

$$\mathtt{M}\ \mathtt{and}\ \mathtt{N} \doteq \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ \mathtt{1}\ )\ \mathtt{else}\ \mathtt{1}$$

$$\mathtt{M}\ \mathtt{or}\ \mathtt{N} \doteq \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ \mathtt{1}\ )$$

It is worth noticing that due to the presence of the $(\mathbf{B}E)$ rule, the following rules with an additive management of contexts are derivable in $\mathrm{STA}_{\mathbf{B}}$:

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N} : \mathbf{B}}{\Gamma \vdash \mathtt{M}\ \mathtt{and}\ \mathtt{N} : \mathbf{B}} \qquad \frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N} : \mathbf{B}}{\Gamma \vdash \mathtt{M}\ \mathtt{or}\ \mathtt{N} : \mathbf{B}}$$

Moreover, there is a term `not` defining the expected boolean function.

ATMs Configurations

The encoding of Deterministic Turing Machine configuration given in [Gaboardi and Ronchi Della Rocca 2007] can be adapted in order to encode Alternating Turing Machine configurations. In fact, an ATM configuration can be viewed as a DTM configuration with an extra information about the state. There are four kinds of state: *accepting* (`A`), *rejecting* (`R`), *universal* ($\wedge$) and *existential* ($\vee$) . We can encode such information by tensor pairs of booleans. In particular:

$\langle 1,0\rangle$	A	$\langle 1,1\rangle$	R	$\langle 0,1\rangle$	$\wedge$	$\langle 0,0\rangle$	$\vee$

We say that 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:

$$\lambda\mathtt{c}.\langle \mathtt{cb}^l_0 \circ \cdots \circ \mathtt{cb}^l_n, \mathtt{cb}^r_0 \circ \cdots \circ \mathtt{cb}^r_m, \langle \mathtt{Q}, \mathtt{k}\rangle\rangle$$

where $\mathtt{cb}^l_0 \circ \ldots \circ \mathtt{cb}^l_n$ and $\mathtt{cb}^r_0 \circ \ldots \circ \mathtt{cb}^r_n$ 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 $\mathtt{k} \equiv \langle \mathtt{k}_1, \mathtt{k}_2\rangle$ is the tensor pair encoding the kind of the state. By convention, the left part of the tape is represented in a reversed order, the alphabet is composed by the two symbols $\mathtt{0}$ and $\mathtt{1}$, the scanned symbol is the first symbol in the right part and final states are divided in accepting and rejecting. In fact, a standard three symbols alphabet (0,1,blank) for the machine working tape can be mapped to our two symbols alphabet $\{\mathtt{0}, \mathtt{1}\}$ in a standard way. For instance we can consider an encoding function ξ that maps each symbol of the ternary alphabet to a pair of symbols of the binary alphabet, e.g. $\xi(0) = (\mathtt{0}, \mathtt{0})$; $\xi(1) = (\mathtt{0}, \mathtt{1})$ and $\xi(\text{blank}) = (\mathtt{1}, \mathtt{1})$. Moreover, we can assume that the transition function δ and the polynomial bounds are designed in accordance with this assumption.
Analogously to what happens in the case of Church numerals, the terms representing configurations presented above can be typed by using indexed types (for every $i \geq 1$) as:

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

Note that the q in $\mathbf{ATM}^q_i$ is the length of the tuple encoding the state. Such a length is determined once a particular ATM is fixed. We need some terms defining operations on ATM. In particular, the term $\mathtt{Init}^m \doteq \lambda\mathtt{t}.\lambda\mathtt{c}.\langle\lambda\mathtt{z}.\mathtt{z}, \lambda\mathtt{z}.\mathtt{t}(\mathtt{c}\mathbf{0})\mathtt{z}, \langle\underline{\mathtt{Q}_0}, \underline{\mathtt{k}_0}\rangle\rangle$ defines the initialization function that takes in input a Church numeral $\underline{\mathtt{n}}$ and gives as output a Turing machine with tape of length n filled by $\mathbf{0}$'s in the initial state $\mathtt{Q}_0 \equiv \langle \mathtt{q}_0, \ldots, \mathtt{q}_m\rangle$ of kind $\mathtt{k}_0 \equiv \langle \mathtt{k}'_0, \mathtt{k}''_0\rangle$ and with the head at the beginning of the tape. It is easy to verify that $\mathtt{Init}^m : \mathbf{N}_i \multimap \mathbf{ATM}^m_i$ for every $i \geq 1$. Note that the term $\mathtt{Init}^m$ and its typing are parametric in the length m of the tuple encoding the state.
An ATM transition relation δ can be considered as the union of the transition functions $\delta_1, \ldots, \delta_n$ being its components. So, we need to show that these components are definable. We decompose an ATM transition function step in two stages. In the first stage, the ATM configuration is decomposed to extract the information needed by the transition relation δ. In the second one, the previously obtained information are combined, depending on the considered transition component δ_j, in order to build the new ATM configuration. The term performing the decomposition stage is:

$$\begin{aligned}&\mathtt{Dec} \doteq \lambda\mathtt{s}.\lambda\mathtt{c}.\mathbf{let}\ \mathtt{s}(\mathtt{F}[\mathtt{c}])\ \mathbf{be}\ \mathtt{l}, \mathtt{r}, \mathtt{p}\ \mathbf{in}\ \mathbf{let}\ \mathtt{p}\ \mathbf{be}\ \mathtt{q}, \mathtt{k}\ \mathbf{in}\ \mathbf{let}\ \mathtt{l}\langle\mathtt{I}, \lambda\mathtt{x}.\mathtt{I}, \mathbf{0}\rangle\\ &\quad \mathbf{be}\ \mathtt{t}_l, \mathtt{c}_l, \mathtt{b}^l_0\ \mathbf{in}\ \mathbf{let}\ \mathtt{r}\langle\mathtt{I}, \lambda\mathtt{x}.\mathtt{I}, \mathbf{0}\rangle\ \mathbf{be}\ \mathtt{t}_r, \mathtt{c}_r, \mathtt{b}^r_0\ \mathbf{in}\ \langle\mathtt{t}_l, \mathtt{t}_r, \mathtt{c}_l, \mathtt{b}^l_0, \mathtt{c}_r, \mathtt{b}^r_0, \mathtt{q}, \mathtt{k}\rangle\end{aligned}$$

where $\mathtt{F}[\mathtt{c}] \doteq \lambda\mathtt{b}.\lambda\mathtt{z}.\mathbf{let}\ \mathtt{z}\ \mathbf{be}\ \mathtt{g}, \mathtt{h}, \mathtt{i}\ \mathbf{in}\ \langle\mathtt{hi} \circ \mathtt{g}, \mathtt{c}, \mathtt{b}\rangle$. It is boring but easy to check that the term $\mathtt{Dec}$ can be typed as $\vdash \mathtt{Dec} : \mathbf{ATM}^q_i \multimap \mathbf{ID}^q_i$, where the indexed type $\mathbf{ID}^q_i$ is used to type the intermediate configuration decomposition and it is defined as $\mathbf{ID}^q_i \doteq \forall\alpha.!^i(\mathbf{B} \multimap \alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha)^2 \otimes ((\mathbf{B} \multimap \alpha \multimap \alpha) \otimes \mathbf{B})^2 \otimes \mathbf{B}^q \otimes \mathbf{B}^2)$. The behaviour of $\mathtt{Dec}$ is the following:

$$\begin{aligned}&\mathtt{Dec}\ (\lambda\mathtt{c}.\langle\mathtt{cb}^l_0 \circ \cdots \circ \mathtt{cb}^l_n, \mathtt{cb}^r_0 \circ \cdots \circ \mathtt{cb}^r_m, \langle\mathtt{Q}, \mathtt{k}\rangle\rangle) \to^*_{\beta\delta}\\ &\qquad\qquad \lambda\mathtt{c}.\langle\mathtt{cb}^l_1 \circ \cdots \circ \mathtt{cb}^l_n, \mathtt{cb}^r_1 \circ \cdots \circ \mathtt{cb}^r_m, \mathtt{c}, \mathtt{b}^l_0, \mathtt{c}, \mathtt{b}^r_0, \mathtt{Q}, \mathtt{k}\rangle\end{aligned}$$

The transition combination stage is performed by the term

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

where $\mathtt{R} \doteq \lambda\mathtt{b}'.\lambda\mathtt{q}'.\lambda\mathtt{k}'.\lambda\mathtt{s}.\mathbf{let}\ \mathtt{s}\ \mathbf{be}\ \mathtt{l}, \mathtt{r}, \mathtt{c}_l, \mathtt{b}_l, \mathtt{c}_r\ \mathbf{in}\ \langle\mathtt{c}_r\mathtt{b}' \circ \mathtt{c}_l\mathtt{b}_l \circ \mathtt{l}, \mathtt{r}, \langle\mathtt{q}', \mathtt{k}'\rangle\rangle$, $\mathtt{L} \doteq \lambda\mathtt{b}'.\lambda\mathtt{q}'.\lambda\mathtt{k}'.\lambda\mathtt{s}.\mathbf{let}\ \mathtt{s}\ \mathbf{be}\ \mathtt{l}, \mathtt{r}, \mathtt{c}_l, \mathtt{b}_l, \mathtt{c}_r\ \mathbf{in}\ \langle\mathtt{l}, \mathtt{c}_l\mathtt{b}_l \circ \mathtt{c}_r\mathtt{b}' \circ \mathtt{r}, \langle\mathtt{q}', \mathtt{k}'\rangle\rangle$ and $\underline{\delta}_j$ is a term defining the δ_j component of the transition relation δ. The term $\texttt{Com}_j$ can be typed as $\vdash \texttt{Com}_j : \mathbf{ID}^q_i \multimap \mathbf{ATM}^q_i$. It combines the symbols obtained after the decomposition stage depending on the considered component δ_j and returns the new ATM configuration. If $\delta_j(\texttt{b}^r_0, \texttt{Q}, \texttt{k}) = (\texttt{b}', \texttt{Q}', \texttt{k}', \text{Right})$, then

$$\begin{aligned}\texttt{Com}_j\ (\lambda\texttt{c}.\langle \texttt{cb}^l_1 \circ \cdots \circ \texttt{cb}^l_n, \texttt{cb}^r_1 \circ \cdots \circ \texttt{cb}^r_m, \texttt{c}, \texttt{b}^l_0, \texttt{c}, \texttt{b}^r_0, \langle \texttt{Q}, \texttt{k}\rangle\rangle) &\to^*_{\beta\delta} \\ \lambda\texttt{c}.\langle \texttt{cb}' \circ \texttt{cb}^l_0 \circ \texttt{cb}^l_1 \circ \cdots \circ \texttt{cb}^l_n, \texttt{cb}^r_1 \circ \cdots \circ \texttt{cb}^r_m, \langle \texttt{Q}', \texttt{k}'\rangle\rangle\end{aligned}$$

otherwise, if $\delta_j(\texttt{b}^r_0, \texttt{Q}, \texttt{k}) = (\texttt{b}', \texttt{Q}', \texttt{k}', \text{Left})$ then

$$\begin{aligned}\texttt{Com}_j\ (\lambda\texttt{c}.\langle \texttt{cb}^l_1 \circ \cdots \circ \texttt{cb}^l_n, \texttt{cb}^r_1 \circ \cdots \circ \texttt{cb}^r_m, \texttt{c}, \texttt{b}^l_0, \texttt{c}, \texttt{b}^r_0, \langle \texttt{Q}, \texttt{k}\rangle\rangle) &\to^*_{\beta\delta} \\ \to^*_{\beta\delta} \lambda\texttt{c}.\langle \texttt{cb}^l_1 \circ \cdots \circ \texttt{cb}^l_n, \texttt{cb}^l_0 \circ \texttt{cb}' \circ \texttt{cb}^r_1 \circ \cdots \circ \texttt{cb}^r_m, \langle \texttt{Q}', \texttt{k}'\rangle\rangle\end{aligned}$$

By combining $\texttt{Dec}$ with $\texttt{Com}_j$ we obtain the j-th component of transition between ATM configurations:

$$\vdash \texttt{Tr}_j = \texttt{Com}_j \circ \texttt{Dec} : \mathbf{ATM}^q_i \multimap \mathbf{ATM}^q_i$$

We can give a term $\texttt{In}$ defining the function that, when supplied by a boolean string and an ATM configuration, writes the input string on the tape of the ATM. This function should consider the encoding function ξ in order to map the input symbols to their concrete instances on the working tape. Given a term $\underline{\xi}$ for the encoding function ξ we can define the desired term $\texttt{In}$ as:

$$\texttt{In} \doteq \lambda\texttt{s}.\lambda\texttt{m}.\texttt{s}(\lambda\texttt{b}.\textbf{let } \underline{\xi}\texttt{b} \textbf{ be } \texttt{b}_1, \texttt{b}_2 \textbf{ in } (\texttt{Tb}_2) \circ \texttt{Dec} \circ (\texttt{Tb}_1) \circ \texttt{Dec})\texttt{m}$$

where $\texttt{T} \doteq \lambda\texttt{b}.\lambda\texttt{s}.\lambda\texttt{c}.\textbf{let } \texttt{sc} \textbf{ be } \texttt{l}, \texttt{r}, \texttt{c}_l, \texttt{b}_l, \texttt{c}_r, \texttt{b}_r, \texttt{q}, \texttt{k} \textbf{ in } \langle \texttt{c}_r\texttt{b} \circ \texttt{c}_l\texttt{b}_l \circ \texttt{l}, \texttt{r}, \langle \texttt{q}, \texttt{k}\rangle\rangle$. This is typable as

$$\vdash \texttt{In} : \mathbf{S} \multimap \mathbf{ATM}^q_i \multimap \mathbf{ATM}^q_i$$

Finally, the term that takes a configuration and return its kind is:

$$\texttt{Kind} \doteq \lambda\texttt{x}.\textbf{let } \texttt{x}(\lambda\texttt{b}.\lambda\texttt{y}.\texttt{y}) \textbf{ be } \texttt{l}, \texttt{r}, \texttt{s} \textbf{ in } (\textbf{let } \texttt{s} \textbf{ be } \texttt{q}, \texttt{k} \textbf{ in } \texttt{k})$$

which is typable as $\vdash \texttt{Kind} : \mathbf{ATM}^{\mathrm{q}}_{\mathrm{i}} \multimap \mathbf{B}^2$.

Evaluation function

Given an ATM $\mathcal{M}$ working in polynomial time we define a recursive evaluation procedure $\texttt{eval}_\mathcal{M}$ that takes a string $\texttt{s}$ and returns $\texttt{0}$ or $\texttt{1}$ if the initial configuration (with the tape filled with $\texttt{s}$) leads to an accepting or rejecting configuration respectively. Without loss of generality we consider ATMs with transition relation δ of degree two. So in particular, at each step we have two transitions terms $\texttt{Tr}^1_\mathcal{M}$ and $\texttt{Tr}^2_\mathcal{M}$ defining the two components δ_1 and δ_2 of the transition relation of $\mathcal{M}$. We need to define some auxiliary functions. In particular, we need a function α acting on states as

$$\begin{array}{ll}\alpha(\texttt{A}, \texttt{M}_1, \texttt{M}_2) = \texttt{A} & \alpha(\wedge, \texttt{M}_1, \texttt{M}_2) = \texttt{M}_1 \wedge \texttt{M}_2 \\ \alpha(\texttt{R}, \texttt{M}_1, \texttt{M}_2) = \texttt{R} & \alpha(\vee, \texttt{M}_1, \texttt{M}_2) = \texttt{M}_1 \vee \texttt{M}_2\end{array}$$

where $\wedge$ and $\vee$ denotes the universal and existential connectives on states. That is, $\texttt{M}_1 \wedge \texttt{M}_2$ is an accepting state only if both the arguments are accepting, $\texttt{M}_1 \vee \texttt{M}_2$ is an accepting state if at least one argument is accepting. This can be defined by the term

$$\begin{aligned}&\alpha(\texttt{M}_0, \texttt{M}_1, \texttt{M}_2) \doteq \textbf{let } \texttt{M}_0 \textbf{ be } \texttt{a}_1, \texttt{a}_2 \textbf{ in } \texttt{ if } \texttt{a}_1 \texttt{ then } (\texttt{ if } \texttt{a}_2 \texttt{ then } \langle \texttt{a}_1, \\ &\pi_2(\texttt{M}_1) \texttt{ or } \pi_2(\texttt{M}_2)\rangle \texttt{ else } \langle \texttt{a}_1, \pi_2(\texttt{M}_1) \texttt{ and } \pi_2(\texttt{M}_2)\rangle) \texttt{ else } \langle \texttt{a}_1, \texttt{a}_2\rangle\end{aligned}$$

It is worth noticing that α has typing:

$$\frac{\Gamma \vdash \mathtt{M}_0 : \mathbf{B}^2 \quad \Gamma \vdash \mathtt{M}_1 : \mathbf{B}^2 \quad \Gamma \vdash \mathtt{M}_2 : \mathbf{B}^2}{\Gamma \vdash \alpha(\mathtt{M}_0, \mathtt{M}_1, \mathtt{M}_2) : \mathbf{B}^2}$$

where the contexts management is additive. This is one of the main reason for introducing the `if` rule with an additive management of contexts. Moreover, note that we do not need any modality here, in particular this means that the α function can be defined in the linear fragment of the $\mathrm{STA}_{\mathbf{B}}$ system.
The evaluation function $\mathtt{eval}_{\mathcal{M}}$ can now be defined as an iteration of an higher order $\mathtt{Step}_{\mathcal{M}}$ function over a `Base` case. Let $\mathtt{Tr}^1_{\mathcal{M}}$ and $\mathtt{Tr}^2_{\mathcal{M}}$ be two closed terms defining the two components of the transition relation. Let us define

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

It is easy to verify that such terms are typable as:

$$\begin{aligned} &\vdash \mathtt{Base} : \mathbf{ATM}^q_i \multimap \mathbf{B}^2 \\ &\vdash \mathtt{Step}_{\mathcal{M}} : (\mathbf{ATM}^q_i \multimap \mathbf{B}^2) \multimap \mathbf{ATM}^q_i \multimap \mathbf{B}^2 \end{aligned}$$

Let P be a polynomial definable by a term `P` typable as $\vdash \mathtt{P} :!^{deg(P)}\mathbf{N} \multimap \mathbf{N}_{2deg(P)+1}$. Then, the evaluation function of an ATM $\mathcal{M}$ working in polynomial time P is definable by the term:

$$\mathtt{eval}_{\mathcal{M}} \doteq \lambda \mathtt{s}.\mathbf{let}\ (\mathtt{P}\ (\mathtt{len}\ \mathtt{s})\ \mathtt{Step}_{\mathcal{M}}\ \mathtt{Base})(\mathtt{In}\ \mathtt{s}\ (\mathtt{Init}^{\mathtt{q}}_{\mathcal{M}}\ (\mathtt{P}\ (\mathtt{len}\ \mathtt{s})))) \ \mathbf{be}\ \mathtt{l}, \mathtt{r}\ \mathbf{in}\ \mathtt{r}$$

which is typable in $\mathrm{STA}_{\mathbf{B}}$ as $\vdash \mathtt{eval}_{\mathcal{M}} :!^t\mathbf{S} \multimap \mathbf{B}$ where $t = \max(deg(P), 1) + 1$.
Here, the evaluation is performed by a higher order iteration, which represents a recurrence with parameter substitutions. Note that by considering an ATM $\mathcal{M}$ that decides a language $\mathcal{L}$, we have that the final configuration is either accepting or rejecting.

LEMMA 5.3. *A decision problem* $\mathcal{D} : \{0,1\}^* \to \{0,1\}$ *decidable by an ATM* $\mathcal{M}$ *in polynomial time is* $\mathbf{B}$*-programmable in* $\mathrm{STA}_{\mathbf{B}}$.

PROOF. $\mathcal{D}(s) = b \iff \mathtt{eval}_{\mathcal{M}}\mathtt{s} \Downarrow \mathtt{b}$ □

From the well known result of [Chandra et al. 1981] we can conclude.

THEOREM 5.4 (POLYNOMIAL SPACE COMPLETENESS). *Every decision problem* $\mathcal{D} \in$ PSPACE *is* $\mathbf{B}$*-programmable in* $\mathrm{STA}_{\mathbf{B}}$.

6. CONCLUSION AND RELATED TOPICS

In this paper we have designed $\mathrm{STA}_{\mathbf{B}}$, a language correct and complete with respect to the polynomial space computations. Namely, the calculus is an extension of λ-calculus, and we supplied a type assignment system for it, such that well typed programs (closed terms of constant type) can be evaluated in polynomial space and moreover all polynomial space decision functions can be computed by well typed programs. In order to perform the complexity bounded evaluation a suitable evaluation machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ has been defined, evaluating programs according to the left-most outer-most evaluation strategy and using two memory devices, one in order to make the evaluation space-efficient and the other in order to avoid backtracking.

We conclude by suggesting some further directions that could bring to new interesting results.

Alternation Revisited

The results presented in this paper have been obtained by exploiting the equivalence [Chandra et al. 1981]:

$$\mathrm{PSPACE} = \mathrm{APTIME}$$

Indeed, evaluations in the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ can be regarded as computations in Alternating Turing Machines. Moreover, the simulation of big step evaluations by means of small step reductions is a reminiscence of the simulation of ATM by means of Deterministic Turing Machines. Conversely, the PSPACE completeness is shown by encoding polynomial time ATM by means of well typed terms. In our completeness proof, as usual in light logics, the modal part of the $\mathrm{STA}_{\mathbf{B}}$ system is only involved in the polynomial iteration (the *quantitative* part), while the ATM behaviour (i.e. the α function) can be defined in the modal free fragment of the system (the *qualitative* part). This suggests that our approach could be further refined in order to revisit some classical complexity results relating time and space [Stockmeyer 1976].

FPSPACE characterization.

FPSPACE is the class of function computable in polynomial space. The completeness for FPSPACE can be obtained by replacing booleans by words over booleans. In particular we can add to STA the type $\mathbf{W}$ and the following rules:

$$\frac{}{\vdash \epsilon : \mathbf{W}} \quad \frac{\Gamma \vdash \mathtt{M} : \mathbf{W}}{\Gamma \vdash \mathbf{0}(\mathtt{M}) : \mathbf{W}} \quad \frac{\Gamma \vdash \mathtt{M} : \mathbf{W}}{\Gamma \vdash \mathbf{1}(\mathtt{M}) : \mathbf{W}} \quad \frac{\Gamma \vdash \mathtt{M} : \mathbf{W}}{\Gamma \vdash \mathbf{p}(\mathtt{M}) : \mathbf{W}}$$

and the conditional

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{W} \quad \Gamma \vdash \mathtt{N}_\epsilon : \mathbf{W} \quad \Gamma \vdash \mathtt{N}_{\mathbf{0}} : \mathbf{W} \quad \Gamma \vdash \mathtt{N}_{\mathbf{1}} : \mathbf{W}}{\Gamma \vdash \mathbf{D}(\mathtt{M}, \mathtt{N}_\epsilon, \mathtt{N}_{\mathbf{0}}, \mathtt{N}_{\mathbf{1}}) : \mathbf{W}}$$

The obtained system $\mathrm{STA}_{\mathbf{W}}$ equipped with the obvious reduction relation could be shown to be FPSPACE sound following what we have done for $\mathrm{STA}_{\mathbf{B}}$. Moreover, analogously to [Leivant and Marion 1993], completeness for FPSPACE can be proved by considering two distinct data types $\mathbf{S}$ (Church representations of Strings) and $\mathbf{W}$ (Flat words over Booleans) as input and output data type respectively.

$\mathrm{STA}_{\mathbf{B}}$ and Soft Linear Logic.

STA has been introduced as a type assignment counterpart of Soft Linear Logic [Lafont 2004]. $\mathrm{STA}_{\mathbf{B}}$ is an extension of STA by booleans constants. One would wonder to exploit the proofs-as-programs correspondence in the design of a purely logical characterization of the class PSPACE.
We can add to SLL (or define by means of second order quantifier) the additive disjunction $\oplus$ and the rules to deal with it, which in a natural deduction style are:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ (\oplus_l I) \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\ (\oplus_r I) \quad \frac{\Gamma \vdash A \oplus B \quad \Delta, A \vdash C \quad \Delta, B \vdash C}{\Gamma, \Delta \vdash C}\ (\oplus E)$$

We can so define $\mathbf{B} = \mathbf{1} \oplus \mathbf{1}$, where $\mathbf{1}$ is the multiplicative unit, and specialize the above rules to booleans:

$$\frac{\Gamma \vdash \mathbf{1}}{\Gamma \vdash \mathbf{1} \oplus \mathbf{1}}\ (0) \quad \frac{\Gamma \vdash \mathbf{1}}{\Gamma \vdash \mathbf{1} \oplus \mathbf{1}}\ (1) \quad \frac{\Gamma \vdash \mathbf{1} \oplus \mathbf{1} \quad \Delta, \mathbf{1} \vdash C \quad \Delta, \mathbf{1} \vdash C}{\Gamma, \Delta \vdash C}\ (\mathbf{E})$$

It is worth noticing that such rules do not change the complexity of SLL. In fact it is essential in order to obtain a logical system behaving as $\mathrm{STA}_{\mathbf{B}}$ to modify the above elimination rule allowing free contraction between contexts Γ and Δ. Hence we can modify the $(\mathbf{E})$ rule as follows:

$$\frac{\Gamma \vdash \mathbf{1} \oplus \mathbf{1} \quad \Gamma, \mathbf{1} \vdash C \quad \Gamma, \mathbf{1} \vdash C}{\Gamma \vdash C} \; (\mathbf{E}_1)$$

The logical system obtained by adding the above modified rule to SLL behaves like $\mathrm{STA}_{\mathbf{B}}$, however, to prove the polynomial space soundness one needs to mimic in the normalization process the abstract machine mechanism of Section 3. Since natural deduction does not permit a fine control of the normalization, one would understand instead how to do this in sequent calculus or proof nets, the two proof formalisms most natural for linear logic. Unfortunately, the logical sequent calculus system obtained by forgetting terms is unsatisfactory. Indeed, the rule $(\mathbf{B}E)$ or $(\mathbf{E}_1)$ has no direct correspondent. Moreover, it looks not so easy to understand how to transfer the complexity bound from the term evaluation to the cut-elimination in a logic. All these difficulties suggest that exploring this direction could be a true test for the light logics principles.

ACKNOWLEDGMENTS

We warmly thank the anonymous referees. Their valuable comments and suggestions helped improve the paper. The first author also thanks Ugo Dal Lago for a discussion that has brought to a nice simplification of Section 4.

REFERENCES

ABITEBOUL, S. AND VIANU, V. 1989. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 71–79.

ASPERTI, A. AND ROVERSI, L. 2002. Intuitionistic light affine logic. *ACM Transactions on Computational Logic 3(1)*, 137–175.

BAILLOT, P., GABOARDI, M., AND MOGBIL, V. 2010. A polytime functional language from light linear logic. In *Proceedings of the 19th European Symposium On Programming*. LNCS, vol. 6012. Springer, 104–124.

BAILLOT, P. AND TERUI, K. 2004. Light types for polynomial time computation in lambda-calculus. In *Proceedings of the 19th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 266–275.

BAILLOT, P. AND TERUI, K. 2009. Light types for polynomial time computation in lambda calculus. *Information and Computation 207,* 1, 41–62.

BARENDREGT, H. 1984. *The Lambda Calculus: Its Syntax and Semantics*, Revised ed. Elsevier/North-Holland, Amsterdam, London, New York.

BELLANTONI, S., NIGGL, K.-H., AND SCHWICHTENBERG, H. 2000. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic 104*, 17–30.

CHANDRA, A. K., KOZEN, D. C., AND STOCKMEYER, L. J. 1981. Alternation. *Journal of the ACM 28,* 1, 114–133.

COPPOLA, P., DAL LAGO, U., AND RONCHI DELLA ROCCA, S. 2005. Elementary affine logic and the call by value lambda calculus. In *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications*. LNCS, vol. 3461. Springer, 131–145.

COPPOLA, P., DAL LAGO, U., AND RONCHI DELLA ROCCA, S. 2008. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science 4,* 4.

DAL LAGO, U. AND SCHÖPP, U. 2010. Functional programming in sublinear space. In In *Proceedings of the 19th European Symposium On Programming*. LNCS, vol. 6012. Springer, 205–225.

DANNER, N. AND ROYER, J. S. 2006. Adventures in time and space. In *Proceedings of the 33th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. 168–179.

GABOARDI, M. 2007. Linearity: an analytic tool in the study of complexity and semantics of programming languages. Ph.D. thesis, Università degli Studi di Torino - Institut National Polytechnique de Lorraine.

GABOARDI, M., MARION, J.-Y., AND RONCHI DELLA ROCCA, S. 2008a. A logical account of PSPACE. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 121–131.

GABOARDI, M., MARION, J.-Y., AND RONCHI DELLA ROCCA, S. 2008b. Soft linear logic and polynomial complexity classes. In *Proceedings of the 2nd Workshop on Logical and Semantic Frameworks, with Applications*. ENTCS, vol. 205. Elsevier, 67–87.

GABOARDI, M. AND RONCHI DELLA ROCCA, S. 2007. A soft type assignment system for λ-calculus. In *Proceedings of the 16th Conference in Computer Science Logic*. LNCS, vol. 4646. Springer, 253–267.

GABOARDI, M. AND RONCHI DELLA ROCCA, S. 2009. From light logics to type assignements: a case study. *Logic Journal of the IGPL, Special Issue on LSFA 2007 17*, 499 – 530.

GIRARD, J.-Y. 1972. Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse de doctorat d'état, Université Paris VII.

GIRARD, J.-Y. 1998. Light linear logic. *Information and Computation 143(2)*, 175–204.

GOERDT, A. 1992. Characterizing complexity classes by higher type primitive recursive definitions. *Theoretical Computer Science 100,* 1, 45–66.

GRÄDEL, E., KOLAITIS, P., LIBKIN, L., MARX, M., SPENCER, J., VARDI, M., VENEMA, Y., AND WEINSTEIN, S. 2007. *Finite Model Theory and its applications*. Springer.

HOFMANN, M. 1999. Linear types and non-size-increasing polynomial time computation. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 464–473.

HOFMANN, M. 2003. Linear types and non-size-increasing polynomial time computation. *Information and Computation 183,* 1, 57–85.

JONES, N. 2001. The expressive power of higher-order types or, life without cons. *Journal of Functional Programming 11,* 1, 55–94.

KAHN, G. 1987. Natural semantics. In *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science*. LNCS, vol. 247. Springer-Verlag, 22–39.

KRIVINE, J.-L. 2007. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation 20,* 3, 199–207.

LAFONT, Y. 2004. Soft linear logic and polynomial time. *Theoretical Computer Science 318,* 1-2, 163–180.

LEIVANT, D. AND MARION, J.-Y. 1993. Lambda calculus characterizations of poly-time. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications*. LNCS, vol. 664. Springer, 274–288.

LEIVANT, D. AND MARION, J.-Y. 1994. Ramified recurrence and computational complexity II: Substitution and poly-space. In *Proceedings of the 8th Conference in Computer Science Logic*. LNCS, vol. 933. Springer, 486–500.

LEIVANT, D. AND MARION, J.-Y. 1997. Predicative functional recurrence and poly-space. In *Theory and Practice of Software Development*. LNCS, vol. 1214. Springer-Verlag, 369–380.

MAUREL, F. 2003. Nondeterministic light logics and NP-time. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, LNCS, vol. 2701. Springer, 241–255.

OITAVEM, I. 2001. Implicit characterizations of pspace. In *Proceedings of the Seminar on Proof Theory in Computer Science*. LNCS, vol. 2183. Springer, 170–190.

OITAVEM, I. 2008. Characterizing pspace with pointers. *Mathematical Logic Quarterly 54,* 3, 323–329.

PLOTKIN, G. D. 2004. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming 60-61*, 17–139. First appeared as DAIMI FN–19 technical report Aarhus University in 1981.

SAVITCH, W. J. 1970. Relationship between nondeterministic and deterministic tape classes. *JCSS 4*, 177–192.

SCHÖPP, U. 2006. Space-efficient computation by interaction. In *Proceedings of the 15th Conference in Computer Science Logic*, LNCS, vol. 4207. Springer, 606–621.

SCHÖPP, U. 2007. Stratified bounded affine logic for logarithmic space. In *Proceedings of the 22nd Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 411–420.

SCHUBERT, A. 2001. The complexity of beta-reduction in low orders. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications*. LNCS, vol. 2044, 400–414.

STOCKMEYER, L. J. 1976. The polynomial-time hierarchy. *Theoretical Computer Science 3,* 1, 1–22.

TERUI, K. 2000. Linear logical characterization of polyspace functions (extended abstract). Unpublished. Presented at the *Second Workshop on Implicit Computational Complexity*.

VARDI, M. 1982. Complexity and relational query languages. In *14th Symposium on Theory of Computing*. ACM, New York, 137–146.

Received June 2010; revised ; accepted January 2011