An Implicit Characterization of PSPACE

Marco Gaboardi
Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna - Mura Anteo Zamboni 7, 40127 Bologna, Italy, gaboardi@cs.unibo.it
and
Jean-Yves Marion
Nancy-University, ENSMN-INPL, Loria B.P. 239, 54506 Vandoeuvre-lès-Nancy, France, Jean-Yves.Marion@loria.fr
and
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.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, Polynomial Space, Linear Logic, Type Assignment, Operational Semantics

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 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 Languages characterizing complexity classes through type assignment systems for λ-calculus are already present in the literature, 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 (EAL) 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 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 In order to use a similar approach for measuring space complexity, since there is no previous logical characterization of PSPACE from which we can start, we exploit 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 ($\mathrm{STA}_{\mathbf{B}}$), 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 obtain 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 SOS evaluation machine, $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$, 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 $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ takes a program (i.e., a closed term well typed with a constant type) as input, then the size of each configuration is polynomially bounded in the size of the input. So every program is evaluated by the 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 one, our work improves on this in the fact that we give a concrete syntactical proof of PSPACE soundness for programs by means of an evaluation machine while Hofmann PSPACE soundness relies on a semantic argument that hides the technical difficulties that one needs to deal with in the evaluation of programs. Moreover, 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. 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 [Gädel 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. 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 $\text{STA}_{\mathbf{B}}$ is introduced and the proofs of subject reduction and strong normalization properties are given. In Section 3 the operational semantics of $\text{STA}_{\mathbf{B}}$ program is defined, through two equivalent abstract evaluation machines. In Section 4 we show that $\text{STA}_{\mathbf{B}}$ programs can be executed in polynomial space. In Section 5 the completeness for PSPACE is proved. 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, $\text{STA}_{\mathbf{B}}$, and we will prove that $\text{STA}_{\mathbf{B}}$ 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. $\text{STA}_{\mathbf{B}}$ 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 1 ($\Lambda_{\mathcal{B}}$).

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

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

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:*

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

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

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

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

(*3*) *The* size *of a term* M *is denoted as* |M| *and is defined inductively as*

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

$$|\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ | = |\mathtt{M}| + |\mathtt{N_0}| + |\mathtt{N_1}| + 1$$

Note that we use the term 0 to denote "true" and the term 1 to denote "false".

NOTATION 1. *Terms are denoted by* M, N, V, 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* λxy.M *stands for* λx.λy.M. *As usual terms are considered up to α-equivalence, namely a bound variable can be renamed provided no free variable is captured. Moreover,* M[N/x] *denotes the capture-free substitution of all free occurrences of* x *in* M *by* N, FV(M) *denotes the set of free variables of* M *and* n_o(x, M) *denotes the number of free occurrences of the variable* x *in* M.

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

DEFINITION 2 (STA$_{\mathbf{B}}$).

(*1*) *The set* $\mathcal{T}_{\mathbf{B}}$ *of* types *is defined as follows:*

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

where α ranges over a countable set of type variables and **B** *is the only* ground type.

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

(*3*) *The system* STA$_{\mathbf{B}}$ *proves judgments of the shape* $\Gamma \vdash \mathtt{M} : \sigma$ *where* Γ *is a context,* 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* dom(Γ) *and* FTV(Γ) *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*

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

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

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

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

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N}_0 : A \quad \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \mathtt{if\ M\ then\ N_0\ else\ N_1} : A}\;(\mathbf{B}E) \qquad \frac{\Gamma \vdash \mathtt{M} : A \quad \alpha \notin \mathrm{FTV}(\Gamma)}{\Gamma \vdash \mathtt{M} : \forall\alpha.A}\;(\forall I)$$

Table I. The Soft Type Assignment system with Booleans

are denoted by Π, Σ, Θ. $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$ *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* $\forall\alpha_1 \ldots \forall\alpha_m.A$, *and* $!^n\sigma$ *is an abbreviation for* $!\ldots!\sigma$ *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 `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 1 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 $\Gamma \vdash \mathtt{M} : \sigma$. □

Moreover, the following property holds:

Lemma 2. $\Gamma, \mathtt{x} : A \vdash \mathtt{M} :!\sigma$ *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 3. *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 or deleting some rules 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 just commutation and erasing of rules, in order to connect the shape of a term with the shape of its typings.

Lemma 3 Generation lemma.

(1) $\Pi \triangleright \Gamma \vdash \lambda\mathtt{x}.\mathtt{M} : \forall\alpha.A$ *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) $\Pi \triangleright \Gamma \vdash \lambda\mathtt{x}.\mathtt{M} : \sigma \multimap A$ *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) $\Pi \triangleright \Gamma \vdash \mathtt{M} :!\sigma$ *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) $\Pi \triangleright !\Gamma \vdash \mathtt{M} :!\sigma$ *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 $\lambda\mathtt{y}.\mathtt{M} \equiv \lambda\mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n]$ and Π ends as:

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

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

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

Then, the desired Π' is:

$$\frac{\dfrac{\Sigma_1 \triangleright \Gamma, \mathtt{x}_1 : \sigma, \ldots, \mathtt{x}_n : \sigma \vdash \lambda\mathtt{y}.\mathtt{N} : A}{\Gamma, \mathtt{x} :!\sigma \vdash \lambda\mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : A}\ (m)}{\Gamma, \mathtt{x} :!\sigma \vdash \lambda\mathtt{y}.\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \forall\alpha.A}\ (\forall I)$$

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:

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

In the case $\mathtt{x}_1, \ldots, \mathtt{x}_n \notin \mathrm{FV}(\mathtt{N})$ 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 $\Delta_1 - \{\mathtt{x}_1 : \tau', \ldots, \mathtt{x}_n : \tau'\}$, then the conclusion follows by the derivation:

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

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, $\mathtt{x}_i \notin \mathrm{FV}(\mathtt{N})$ 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 variable $\mathtt{x}$. The desired derivation Π' is Σ_1 followed by δ', followed by (sp).

□

2.1 Subject reduction

In order to prove subject reduction, we need to prove before that the system enjoys the property of substitution. This last property cannot be proved in a standard way, since the linearity of the axioms and the fact that the rule (m) renames some variables both in the subject and in the context. So, in order to prove that $\Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ and $\Delta \vdash \mathtt{N} : \mu$ $(\Gamma \# \Delta)$ implies $\Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$, we need to consider all the axioms introducing variables which will be renamed as $\mathtt{x}$ in the derivation itself. We need to replace each of them by a disjoint copy of the derivation proving $\Delta \vdash \mathtt{N} : \mu$, and finally to apply a suitable numbers of (m) rules. In order to formalize this procedure we need to introduce the notion of height of a variable in a derivation.

Definition 4. *Let* $\Pi \rhd \Gamma, \mathtt{x} : \tau \vdash \mathtt{M} : \sigma$. *The* height *of* $\mathtt{x}$ *in* Π *is inductively defined as follows:*

—*if the last rule of* Π *is:*

$$\frac{}{\mathtt{x} : A \vdash \mathtt{x} : A}\ (Ax) \quad or \quad \frac{\Gamma' \vdash \mathtt{N} : \sigma}{\Gamma', \mathtt{x} : A \vdash \mathtt{N} : \sigma}\ (w)$$

then the height of $\mathtt{x}$ *in* Π *is* 0.

—*if the last rule of* Π *is:*

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

then the height of $\mathtt{x}$ *in* Π *is the maximum between the heights of* $\mathtt{x}_i$ *in* Σ *for* $1 \leq i \leq k$ *plus one.*

—*If* $\mathtt{x} : \tau \in \Gamma$ *and the last rule of* Π *is*

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

Then the height of $\mathtt{x}$ *in* Π *is the maximum between the heights of* $\mathtt{x}$ *in* Σ, Θ_0 *and* Θ_1 *respectively, plus one.*

—*In every other case there is only one assumption with subject* $\mathtt{x}$ *both in the context of the conclusion of the rule and in the context of one of its premises* Σ*. Then the height of* $\mathtt{x}$ *in* Π *is equal to the height of* $\mathtt{x}$ *in* Σ *plus one.*

We can now prove the substitution lemma.

Lemma 4 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. Let Π and Σ be the derivations proving respectively $\Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ and $\Delta \vdash \mathtt{N} : \mu$. By induction on the height of $\mathtt{x}$ in Π. Base cases (Ax) and (w) are trivial. The cases where Π ends either by $(\multimap I), (\forall I), (\forall E)$ or $(\multimap E)$ follow directly from the induction hypothesis.
Let Π ends by (sp) rule with premise $\Pi' \rhd \Gamma', \mathtt{x} : \mu' \vdash \mathtt{M} : \sigma'$. Then by Lemma 3.3, $\Sigma \rightsquigarrow \Sigma''$ which is composed by a subderivation ending with an (sp) rule with premise $\Sigma' \rhd \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). By induction hypothesis we have a derivation $\Theta' \rhd \Gamma', \Delta' \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma'$. By applying the rule (sp) and the sequence of (w) and/or (m) rules we obtain $\Theta \rhd \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$.
Consider the case Π ends by:

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

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

$$\Gamma, \Delta \vdash \ \mathtt{if}\ \mathtt{M}_0[\mathtt{N}/\mathtt{x}]\ \mathtt{then}\ \mathtt{M}_1[\mathtt{N}/\mathtt{x}]\ \mathtt{else}\ \mathtt{M}_2[\mathtt{N}/\mathtt{x}] \ : A$$

Consider the case Π ends by:

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

By Lemma 3.3 $\Sigma \rightsquigarrow \Sigma''$ ending by an (sp) rule with premise $\Sigma' \rhd \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). Consider fresh copies of the derivation Σ' i.e. $\Sigma'_j \rhd \Delta'_j \vdash \mathtt{N}_j : \mu'$ where $\mathtt{N}_j$ and Δ'_j are fresh copies of $\mathtt{N}$ and Δ' $(1 \leq j \leq m)$.

Let $\mathtt{x}_i$ be such that its height is maximal between the heights of all $\mathtt{x}_j$ $(1 \leq j \leq m)$. By induction hypothesis there is a derivation:

$$\Theta_i \rhd \Gamma, \mathtt{x}_1 : \mu', \ldots, \mathtt{x}_{i-1} : \mu', \mathtt{x}_{i+1} : \mu', \ldots, \mathtt{x}_m : \mu', \Delta'_i \vdash \mathtt{M}[\mathtt{N}_i/\mathtt{x}_i] : \sigma$$

Then, we can repeatedly apply induction hypothesis to obtain a derivation $\Theta' \rhd \Gamma, \Delta'_1, \ldots, \Delta'_m \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \cdots, \mathtt{N}_m/\mathtt{x}_m] : \sigma$. Finally by applying repeatedly the rules (m) and (w) the conclusion follows. $\square$

We can finally prove the main property of this section.

Lemma 5 Subject Reduction.
Let $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \rightarrow_{\beta\delta} \mathtt{N}$. *Then,* $\Gamma \vdash \mathtt{N} : \sigma$.

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

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

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

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

where $\Gamma = \Gamma_1, \Gamma_2$. The others follow directly by induction hypothesis. Clearly $(\lambda \mathtt{x}.\mathtt{M})\mathtt{N} \rightarrow_\beta \mathtt{M}[\mathtt{N}/\mathtt{x}]$. By Lemma 3.2 $\Pi \rightsquigarrow \Pi_1$ ending as

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

By the Substitution Lemma 4 since $\Pi_2 \rhd \Gamma_1, \mathtt{x} : \sigma \vdash \mathtt{M} : A$ and $\Sigma \rhd \Gamma_2 \vdash \mathtt{N} : \sigma$ we have $\Gamma_1, \Gamma_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : A$, hence the conclusion follows. $\square$

It is worth noting 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 1. *Consider for* $n \in \mathbb{N}$ *terms* $\mathtt{M}_n$ *of the shape:*

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

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

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 Fig. II and it differs from the original system since it has explicit rules

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

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

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

Table II. System F with explicit contraction and weakening 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 5. *The types of* System F *are defined by the following grammar:*

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

where α ranges over a countable set of type variables.

We firstly define a forgetful map over types and terms.

Definition 6. *The map $(-)^*$ 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:

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

The following lemma assures that the translation well behaves.

Lemma 6. *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

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

The case Π consists in the $(\mathbf{B}_1 I)$ rule is similar. The case Π ends by (sp) rule follows directly by induction hypothesis. The cases where Π ends either by a $(\multimap I)$, $(\multimap E)$

or (w) rules follow by induction hypothesis and an application of the same rule in System F. In the case Π ends as

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{B} \quad \Gamma \vdash \mathtt{N}_0 : A \quad \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash \ \mathtt{if\ M\ then\ N_0\ else\ N_1} \ : A} \ (\mathbf{B}E)$$

we have a derivation ending as:

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

□

Moreover, the translation preserves the reduction.

Lemma 7 Simulation. *The following diagrams commutes*

$$\begin{tikzcd} \mathtt{M} \arrow[r, "\beta\delta"] \arrow[d, "*"] & \mathtt{N} \arrow[d, "*"] \\ (\mathtt{M})^* \arrow[r, "+"', "\beta"] & (\mathtt{N})^* \end{tikzcd}$$

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

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

the other case is analogous. By definition of the map $(\ )^*$ we have:

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

and clearly:

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

and so the conclusion. □

Now, we have the following.

Theorem 1 Strong Normalization.
If $\Gamma \vdash \mathtt{M} : \sigma$ then $\mathtt{M}$ is strongly normalizing with respect to the relation $\rightarrow_{\beta\delta}$.

Proof. By Lemmas 6 and 7 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 $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$

The machine $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ evaluates programs according to the leftmost outermost strategy. If restricted to λ-calculus, the machine $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ 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 $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ uses two memory devices, the m-context and the **B**-context, that memorize respectively the assignments to variables and the control flow.

Definition 7.

—*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 size of each variable assignment 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* **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 **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 **B***-context* $\mathcal{C}[\circ]$*, denoted by* $\#(\mathcal{C}[\circ])$*, is the number of nested* **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 **B**-contexts $\mathcal{C}[\circ]$ can be seen as a stack of atomic contexts where its 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* **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 8. *Let* $\mathcal{A} = \{\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n := \mathtt{N}_n\}$ *be an 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 9. *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.

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

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

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

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

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

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

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

Remark 1. *It is easy to check that every term has the following shape:* $\lambda\mathtt{x}_1 ... \mathtt{x}_n.\zeta\mathtt{V}_1 \cdots \mathtt{V}_m$, *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:

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

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. They need some comments, we describes the rules bottom-up. The (Ax) rule is obvious. 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 remembered 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 $(\ \mathtt{if}\ \mathtt{0})$ and $(\ \mathtt{if}\ \mathtt{1})$ 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 permits 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 10.

(*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*

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

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

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

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* $\circ, \varepsilon \models \mathtt{M} \Downarrow b$ *then we say that* $\mathtt{M}$ evaluates, *and we write* $\mathtt{M} \Downarrow$. *As usual,* $\models \mathtt{M} \Downarrow b$ *is a short for* $\circ, \varepsilon \models \mathtt{M} \Downarrow b$.

(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 a 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 1.

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 8. *(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 9.

(1) Let $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow b$ *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 $\mathbf{B}$, 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:

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

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

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

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 (`if 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 is 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 $\mathtt{NQV}_1...\mathtt{V}_m$.
Suppose $\mathtt{N} \equiv \lambda\mathtt{x.R}$. There are two cases, either $(\mathtt{M})^{\mathcal{A}} \equiv ((\lambda\mathtt{x.R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}}$ or $(\mathtt{M})^{\mathcal{A}} \equiv (\mathtt{yQ}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}}$ and $\{\mathtt{y} := \lambda\mathtt{x.R}'\} \in \mathcal{A}$.
Let us consider the first case. Then $((\lambda\mathtt{x.R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \to_\beta (\mathtt{R}'[\mathtt{Q}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}@\{\mathtt{x}':=\mathtt{Q}'\}}$. 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 $\{\mathtt{y} := \lambda\mathtt{x.R}'\} \in \mathcal{A}$, then $(\mathtt{yQ}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \to_\beta (\mathtt{R}'[\mathtt{Q}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\mathtt{R}'[\mathtt{x}'/\mathtt{x}]\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}@\{\mathtt{x}':=\mathtt{Q}'\}}$. 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), $[\circ], \mathcal{A} \models (\lambda\mathtt{x.R}')\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$. Finally, by one application of the rule (h), since $\{\mathtt{y} := \lambda\mathtt{x.R}'\} \in \mathcal{A}$, we have $[\circ], \mathcal{A} \models \mathtt{yQ}'\mathtt{V}'_1...\mathtt{V}'_m \Downarrow \mathtt{b}$.
The remaining case is the one where $\mathtt{N} \equiv \texttt{ if } \mathtt{M}' \texttt{ then } \mathtt{N}'_0 \texttt{ else } \mathtt{N}'_1$. By definition $(\mathtt{M})^{\mathcal{A}} \equiv ((\texttt{ if } \mathtt{M}' \texttt{ then } \mathtt{N}'_0 \texttt{ else } \mathtt{N}'_1\ )\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}} \equiv (\texttt{ if } (\mathtt{M}')^{\mathcal{A}} \texttt{ then } (\mathtt{N}'_0)^{\mathcal{A}} \texttt{ else } (\mathtt{N}'_1)^{\mathcal{A}}\ )(\mathtt{Q}')^{\mathcal{A}}(\mathtt{V}'_1)^{\mathcal{A}}...(\mathtt{V}'_m)^{\mathcal{A}}$. 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 that 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 8.1, $(\texttt{ if } [\circ] \texttt{ then } \mathtt{N}'_0 \texttt{ else } \mathtt{N}'_1\ )\mathtt{Q}'\mathtt{V}'_1...\mathtt{V}'_m, \mathcal{A} \models \mathtt{M}' \Downarrow \mathtt{b}'$. 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 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 (`if 0`). The case $(\mathtt{M}')^{\mathcal{A}} = \mathtt{b}'$ is easier. The case $(\mathtt{M})^{\mathcal{A}} \equiv (\mathtt{yQ}'\mathtt{V}'_1...\mathtt{V}'_m)^{\mathcal{A}}$, and $(\mathtt{y} := \texttt{ if } \mathtt{M}' \texttt{ then } \mathtt{N}'_0 \texttt{ else } \mathtt{N}'_1\ )$, is similar, but both rules (h) and (`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 2. *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 9.(1).

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

$$\frac{[\mathtt{x} := \mathtt{N}] \in \overline{\mathcal{S}\cdot\mathcal{A}}}{\langle \mathcal{S},\mathcal{C},\mathcal{A} \succ \mathtt{x}\mathtt{V}_1\cdots\mathtt{V}_m\rangle \mapsto \langle \mathcal{S},\mathcal{C},\mathcal{A} \succ \mathtt{N}\mathtt{V}_1\cdots\mathtt{V}_m\rangle}\ (h)$$

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

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

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

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

Table V. The small step machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$

(2) It follows directly by Lemma 9.(3). □

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

The proof that programs are evaluated by the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ in polynomial space needs 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 we depict a small step abstract machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$ that is able to reduce sequentially programs in $\mathrm{STA}_{\mathbf{B}}$ following a leftmost outermost strategy and that exploit a use of contexts similar to the one implemented by the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. The rules are similar to the ones in Table III but we need a further stack in order to maintain the desired complexity property.
In what follows we show that every big step computation has its small step correspondent. So, the small step machine by making explicit the evaluation order clarifies the fact that every configuration depends uniquely on the previous one (thanks to the use of contexts). From this we can deduce that the space needed in order to evaluate a program is the maximum space used by one of its configurations. The big step machine has the advantage of being more abstract and this makes it easy to prove the complexity properties. In fact, the use of a further stack makes more difficult the proofs of such properties for the small step machine. For this reason in what follows we prefer to work on the big step machine. In order to state formally the behaviour of the machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$ we need to define a further stack containing m-contexts, this is done in the following definition.

Definition 11.

—*An* m-stack $\mathcal{S}$ *is a stack of m-contexts. The symbol* ϵ *denote the empty m-stack. The expression* $\mathcal{S}\cdot\mathcal{A}$ *denotes the operation of pushing the m-context* $\mathcal{A}$ *on the* *m-stack $\mathcal{S}$. The set of m-stacks is denoted by* $\mathrm{Stk_m}$. *The expression* $\overline{\mathcal{S}}$ *denotes the m-context obtained by concatenating all the m-context in* $\mathcal{S}$*, i.e.* $\overline{\epsilon} = \varepsilon$ *and* $\overline{\mathcal{S} \cdot \mathcal{A}} = \overline{\mathcal{S}}@\mathcal{A}$.

—*The* reduction relation $\mapsto \subseteq (\mathrm{Stk_m} \times \mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx_m} \times \Lambda_{\mathcal{B}}) \times (\mathrm{Stk_m} \times \mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx_m} \times \Lambda_{\mathcal{B}})$ *is the relation inductively defined by the rules of* $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$. *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* b *such that* $\langle \epsilon, \circ, \varepsilon \succ \mathtt{M}\rangle \mapsto^* \langle \epsilon, \circ, \mathcal{A} \succ b\rangle$ *for some* $\mathcal{A}$*, then we say that* $\mathtt{M}$ reduces *to* b*, and we simply write* $\mathtt{M} \mapsto^* b$ *for short.*

We now prove that we have a direct correspondence between the configurations of a computation in the big step machine and the small step machine configurations. Given a big step abstract machine derivation $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$, we can define a translation $(-)^{\mathsf{s}}$ assigning to each configuration $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$ a small step abstract machine configuration $\langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ \mathtt{N}\rangle$ such that $\overline{\mathcal{S} \cdot \mathcal{A}'} = \mathcal{A}$. Let $(-)^{\mathsf{s}}$ be inductively defined, for every configuration $\phi \in \nabla$, on the length n of $\mathtt{path}_{\nabla}(\phi)$ as:

—if $n = 1$ then

$$(\circ, \varepsilon \models \mathtt{M} \Downarrow \mathtt{b})^{\mathsf{s}} = \langle \epsilon, \circ, \varepsilon \succ \mathtt{M}\rangle$$

—if $n = m+1$ then for some ψ we have $\mathtt{path}_{\nabla}(\phi) = \mathtt{path}_{\nabla}(\psi)+1$ and in particular we have a rule (R) like the following

$$\frac{\mathcal{C}_1, \mathcal{A}_1 \models \mathtt{N}_1 \Downarrow \mathtt{b}_1 \quad \cdots \quad \mathcal{C}_k, \mathcal{A}_k \models \mathtt{N}_k \Downarrow \mathtt{b}_k}{\psi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}} \ (R)$$

for $1 \leq k \leq 2$ where the length of $\mathtt{path}_{\nabla}(\psi)$ is m and ϕ is one of the premise configurations of (R). We now proceed by case on (R).
If (R) is the rule:

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

Then, by induction hypothesis we have $(\psi)^{\mathsf{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ (\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m\rangle$ such that $\overline{\mathcal{S} \cdot \mathcal{A}'} = \mathcal{A}$, so we can define

$$(\phi)^{\mathsf{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}'@[\mathtt{x}' := \mathtt{N}] \succ \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m\rangle$$

and clearly $\overline{\mathcal{S} \cdot (\mathcal{A}'@\{\mathtt{x}' := \mathtt{N}\})} = \mathcal{A}\{\mathtt{x}' := \mathtt{N}\}$.
If (R) is the rule:

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

Then, by induction hypothesis we have $(\psi)^{\mathsf{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m\rangle$ such that $\overline{\mathcal{S} \cdot \mathcal{A}'} = \mathcal{A}$ so we can define

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

If (R) is the rule:

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

by induction hypothesis $(\psi)^{\mathtt{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ (\texttt{ if M then N}_0 \texttt{ else N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ such that $\overline{\mathcal{S} \cdot \mathcal{A}'} = \mathcal{A}$ and we have two distinct cases. Consider the case that $\phi \succ \mathcal{C}[(\texttt{ if } [\circ] \texttt{ then N}_0 \texttt{ else N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0}$, then we can define

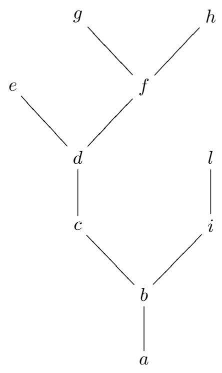
$$(\phi)^{\mathtt{s}} = \langle \mathcal{S} \cdot \mathcal{A}', \mathcal{C}[(\texttt{ if } [\circ] \texttt{ then N}_0 \texttt{ else N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \varepsilon \succ \mathtt{M} \rangle$$

Analogously, if $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}$ then we can define

$$(\phi)^{\mathtt{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ \mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$$

The case (R) is the rule (if 1) is similar.

The translation 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 immediatly follows the node h in the above figure.
Now we can state an important result.

LEMMA 10. *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:*

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

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 possible cases. Consider the case where the rule with conclusion ϕ is (β). Then, we are in a situation as:

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

then

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

where $\mathcal{A} = \overline{\mathcal{S} \cdot \mathcal{A}'}$. 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

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

thanks to the fact that $\{\mathtt{x} := \mathtt{N}\} \in \mathcal{A} = \overline{\mathcal{S} \cdot \mathcal{A}'}$.
If the rule with conclusion ϕ is:

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

then

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

The case of the rule ($\mathtt{if}\ \mathtt{1}$) is analogous.
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 $\psi \in \nabla$ following ϕ such that $\phi \neq \psi$. Otherwise, ∇ has a subderivation $\diamond$ of the shape:

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

where by definition of left-depth-first visit ψ is the configuration following ϕ. Note in particular that $\mathtt{path}_{\diamond}(\phi)$ does not cross any $\mathtt{if}$-rule by following its left premise. In particular, by definition of the translation $(-)^{\mathsf{s}}$ this means that if $(\phi')^{\mathsf{s}} = \langle \mathcal{S}, \mathcal{C}, \mathcal{A}' \succ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle$ then $(\phi)^{\mathsf{s}} = \langle \mathcal{S} \cdot \mathcal{A}', \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A}'' \succ \mathtt{b} \rangle$ for some $\mathcal{A}''$ so in particular we have

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

and the proof is given. □

We can now use this result to prove that computations in the big step machine correspond to computations in the small step one.

THEOREM 3. *Let* $\mathtt{M} \in \mathcal{P}$. *Then:*

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

PROOF. By repeatedly applying Lemma 10. □

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. Indeed, Lemma 10, 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. Moreover, since clearly in the small step machine every step depends only on the previous one, the definition of the translation $(-)^{\mathsf{s}}$ and Lemma 10 imply that also in $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ every execution step depends only on the previous one.

EXAMPLE 2. *By returning to the computation example in Table IV, it is worth noting that to pass from the configuration ϕ to the configuration ψ all necessary information are already present in the configuration ϕ itself. We can view such a step as a $\rightarrow_{\delta}$ step* $(\ \mathtt{if\ 0\ then\ x_1\ else\ x_1}\ )^{\mathcal{A}_3} \rightarrow_{\delta} (\mathtt{x}_1)^{\mathcal{A}_3}$ *noting that* $(\mathtt{x}_1)^{\mathcal{A}_3} \equiv (\mathtt{x}_1)^{\mathcal{A}_2}$.

In fact, the behaviour shown in the above example can be generalized, so in this sense we don't need neither mechanism for backtracking nor the memorization of parts of the computation tree. Using this property, we can define in a similar way the notion of space used to evaluate a term in the two machines. Let us first define the *size* of a configuration in both the machines.

DEFINITION 12.

(1) *If* $\langle \mathcal{S}, \mathcal{C}, \mathcal{A} \succ \mathtt{M} \rangle$ *is a configuration in* $\mathrm{k}_{\mathcal{B}}^{\mathcal{C}}$, *then its* size *is* $|\mathcal{S}| + |\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}|$.

(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 13.

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

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

LEMMA 11. *Let* $\mathtt{M} \in \mathcal{P}$. *Then:*

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

PROOF. By definition of the translation $(-)^{\mathsf{s}}$ and Lemma 10. □

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.

3.3 Space Measures

In this subsection we will connect the space measure of the big step machine with the one of the typable term to be evaluated. In particular, we emphasize the relations between machine computations and type derivations.
In what follows we introduce some relations between the size of the contexts and the behaviour of the machine, which will be useful later.

DEFINITION 14. *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 12. *Let $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$ be a computation. Then, for each configuration $\phi \succ \mathcal{C}_i, \mathcal{A}_i \models \mathtt{P}_i \Downarrow \mathtt{b}' \in \nabla$:*

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

PROOF.

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

The following is a key property for proving soundness.

PROPERTY 1. *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 $\{x_j := \mathtt{N}_j\} \in \mathcal{A}$ then $\mathtt{N}_j$ is an instance (possibly with fresh variables) of a subterm of $\mathtt{M}$.*

PROOF. The property is proven 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 $\mathtt{x}_j := \mathtt{N}_j$, where $\mathtt{N}_j$ 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:

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

If $\mathtt{N}_j$ 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 $(\lambda \mathtt{x}.\mathtt{P})\mathtt{N}_j\mathtt{V}_1 \cdots \mathtt{V}_n$, 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 $\mathtt{N}_j$ 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. □

The next lemma gives upper bounds to the size of the m-context, of the **B**-context and of the subject of a configuration.

LEMMA 13. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$ *then for each configuration* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \Pi$*:*

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

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 12.1 and Property 1.
(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 1.
(3) By inspection of the rules of Table III it is easy to verify that **B**-contexts can grow only by applications of $(\mathtt{if}\ 0)$ and $(\mathtt{if}\ 1)$ rules. So the conclusion follows directly by Lemma 12.2. □

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. In fact, 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. Indeed 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 through derivations with different degrees, nevertheless for each program there is a sort 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 15.

(*1*) *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*. It is inductively defined on the height of* Π *as follows:*
—*if* Π *consists of a* (Ax) *or of a* $(\mathbf{B}_{\mathtt{b}}I)$ *rule then* $\mathtt{d}(\Pi) = 0$

—*if* Π *ends by a rule*

$$\frac{\Sigma}{\Gamma \vdash \mathtt{M} : \sigma}\ (R)$$

where $(R) \in \{(w), (\multimap I), (m), (\forall E), (\forall I)\}$ *then* $\mathtt{d}(\Pi) = \mathtt{d}(\Sigma)$

—*If* Π *ends by a rule*

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

then $\mathtt{d}(\Pi) = \max\{\mathtt{d}(\Sigma), \mathtt{d}(\Theta)\}$

—*If* Π *ends by a rule*

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

then $\mathtt{d}(\Pi) = \max\{\mathtt{d}(\Sigma), \mathtt{d}(\Theta_0), \mathtt{d}(\Theta_1)\}$

—*if* Π *ends by a rule*

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

then $\mathtt{d}(\Pi) = \mathtt{d}(\Sigma) + 1$

(2) *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 $\mathcal{P}_n$. Moreover if $\mathtt{M} \in \mathcal{P}_d$ then $\mathtt{M} \in \mathcal{P}_e$ for every $e \geq d$.

This section is divided into two subsections. In the first, we will prove an intermediate result, namely we will give the notion of space weight of a derivation, and we will prove that the subject reduction does not increase it. Moreover, this result is extended to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. In the second subsection, the soundness with respect to PSPACE will be proved.

4.1 Space and $\mathrm{STA}_{\mathbf{B}}$

We need to define measures of both terms and proofs, which are an adaptation of those given by Lafont in [Lafont 2004].

Definition 16.

—*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* space weight $\delta(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows:*

—*If* Π *consists of a* (Ax) *or of a* $(\mathbf{B}_{\mathtt{b}}I)$ *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 $\delta(\Pi, r) = r\delta(\Sigma, r)$.
—*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*

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

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.*

In order to prove that the subject reduction does not increase the space weight of a derivation, we need to rephrase the Substitution Lemma taking into account this measure.

LEMMA 14 WEIGHTED SUBSTITUTION LEMMA. *Let* $\Pi \rhd \Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ *and* $\Sigma \rhd \Delta \vdash \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$. *There exists* $\Theta \rhd \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$ *such that if* $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta(\Theta, r) \leq \delta(\Pi, r) + \delta(\Sigma, r)$$

PROOF. It suffices to verify how the weights are modified by the proof of Lemma 4. We proceed by induction on the height of $\mathtt{x}$ in Π. Base cases are trivial and in the cases where Π ends by $(\multimap I), (\forall I), (\forall E)$ and $(\multimap E)$ rules the conclusion follows directly by induction hypothesis.
Consider the case Π ends by:

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

Then by Lemma 3.3 $\Sigma \rightsquigarrow \Sigma''$ which is composed by a subderivation ending with an (sp) rule with premise $\Sigma' \rhd \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). By induction hypothesis we have a derivation $\Theta' \rhd \Gamma', \Delta' \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma'$. By applying the rule (sp) and the sequence of (w) and/or (m) rules we obtain $\Theta \rhd \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$. Now, $\delta(\Pi, r) = r\delta(\Pi', r)$ and $\delta(\Sigma, r) = r\delta(\Sigma', r)$. By the induction hypothesis $\delta(\Theta', r) \leq \delta(\Pi', r) + \delta(\Sigma', r)$ and applying (sp):

$$\delta(\Theta, r) \leq r(\delta(\Pi', r) + \delta(\Sigma', r)) = \delta(\Pi, r) + \delta(\Sigma, r)$$

Consider the case Π ends by:

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

Then, by the induction hypothesis there are derivations $\Theta_0 \triangleright \Gamma, \Delta \vdash \mathtt{M}_0[\mathtt{N}/\mathtt{x}] : \mathbf{B}$, $\Theta_1 \triangleright \Gamma, \Delta \vdash \mathtt{M}_1[\mathtt{N}/\mathtt{x}] : A$ and $\Theta_2 \triangleright \Gamma, \Delta \vdash \mathtt{M}_2[\mathtt{N}/\mathtt{x}] : A$ such that $\delta(\Theta_i, r) \leq \delta(\Pi_i, r) + \delta(\Sigma, r)$ for $0 \leq i \leq 2$. By applying a $(\mathbf{B}E)$ rule we obtain a derivation Θ with conclusion:

$$\Gamma, \Delta \vdash \mathtt{if}\ \mathtt{M}_0[\mathtt{N}/\mathtt{x}]\ \mathtt{then}\ \mathtt{M}_1[\mathtt{N}/\mathtt{x}]\ \mathtt{else}\ \mathtt{M}_2[\mathtt{N}/\mathtt{x}]\ : A$$

Since, by definition $\delta(\Pi, r) = \max_{0 \leq i \leq 2}(\delta(\Pi_i, r)) + 1$, we have

$$\delta(\Theta, r) \leq \max_{0 \leq i \leq 2}(\delta(\Pi_i, r) + \delta(\Sigma, r)) + 1 = \max_{0 \leq i \leq 2}(\delta(\Pi_i, r)) + 1 + \delta(\Sigma, r) = \delta(\Pi, r) + \delta(\Sigma, r)$$

Consider the case $\mu \equiv !\mu'$ and Π ends by:

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

By Lemma 3.3, $\Sigma \rightsquigarrow \Sigma''$ ending by an (sp) rule with premise $\Sigma' \triangleright \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). Hence, $\delta(\Sigma, r) = r\delta(\Sigma', r)$. Consider fresh copies of the derivation Σ' i.e. $\Sigma'_j \triangleright \Delta'_j \vdash \mathtt{N}_j : \mu'$ where $\mathtt{N}_j$ and Δ'_j are fresh copies of $\mathtt{N}$ and Δ' respectively, trivially $\delta(\Sigma', r) = \delta(\Sigma'_j, r)$ $(1 \leq j \leq m)$.
Let $\mathtt{x}_i$ be such that its height is maximal between the heights of all $\mathtt{x}_j$ $(1 \leq j \leq m)$. By induction hypothesis there is a derivation:

$$\Theta_i \triangleright \Gamma, \mathtt{x}_1 : \mu', \ldots, \mathtt{x}_{i-1} : \mu', \mathtt{x}_{i+1} : \mu', \ldots, \mathtt{x}_m : \mu', \Delta'_i \vdash \mathtt{M}[\mathtt{N}_i/\mathtt{x}_i] : \sigma$$

and since $\delta(\Pi, r) = \delta(\Pi', r)$, we have $\delta(\Theta_i, r) \leq \delta(\Pi', r) + \delta(\Sigma', r)$. Then, we can repeatedly apply induction hypothesis to obtain a derivation $\Theta' \triangleright \Gamma, \Delta'_1, \ldots, \Delta'_m \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \cdots, \mathtt{N}_m/\mathtt{x}_m] : \sigma$. such that $\delta(\Theta', r) \leq \delta(\Pi', r) + m\delta(\Sigma', r)$ and since $r \geq \mathtt{rk}(\Pi)$ then:

$$\delta(\Theta', r) \leq \delta(\Pi', r) + r\delta(\Sigma', r) = \delta(\Pi, r) + \delta(\Sigma, r)$$

Finally by applying repeatedly the rules (m) and (w) that leave the space weight δ unchanged, the conclusion follows. □

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

Lemma 15. *Let* $\mathtt{P} \in \mathcal{P}$ *and* $\nabla :: \models \mathtt{P} \Downarrow \mathtt{b}$.

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

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

Then, for every derivation $\Sigma \triangleright \vdash ((\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} : \mathbf{B}$ *there exists a derivation* $\Theta \triangleright \vdash (\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@\{\mathtt{x}':=\mathtt{N}\}} : \mathbf{B}$ *such that for every* $r \geq \mathtt{rk}(\Sigma)$*:*

$$\delta(\Sigma, r) > \delta(\Theta, r)$$

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

$$\frac{\mathcal{C}',\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b} \quad \mathcal{C},\mathcal{A} \models \mathtt{N_b V_1 \cdots V}_m \Downarrow \mathtt{b}'}{\mathcal{C},\mathcal{A} \models (\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ )\mathtt{V_1 \cdots V}_m \Downarrow \mathtt{b}'}\ (\ \mathtt{if\ \ b})$$

where $\mathcal{C}' \equiv \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then\ N_0\ else\ N_1}\ )\mathtt{V_1 \cdots V}_m]$*. Then, for each derivation* $\Sigma \triangleright \vdash ((\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ )\mathtt{V_1 \cdots V}_m)^{\mathcal{A}} : \mathbf{B}$ *there are derivations* $\Theta \triangleright \vdash (\mathtt{M})^{\mathcal{A}} : \mathbf{B}$ *and* $\Pi \triangleright \vdash (\mathtt{N_b V_1 \cdots V}_m)^{\mathcal{A}} : \mathbf{B}$ *such that for every* $r \geq \mathtt{rk}(\Sigma)$*:*

$$\delta(\Sigma, r) > \delta(\Theta, r) \quad \textit{and} \quad \delta(\Sigma, r) > \delta(\Pi, r)$$

PROOF.

(1) We proceed by induction on m. Consider the case $m = 0$. We need to prove that if $\Pi \triangleright \Gamma \vdash (\lambda \mathtt{x.M})\mathtt{N} : \sigma$, then there exists $\Pi' \triangleright \Gamma \vdash \mathtt{M[N/x]} : \sigma$ with $\mathtt{rk}(\Pi) \geq \mathtt{rk}(\Pi')$ such that for $r \geq \mathtt{rk}(\Pi)$:

$$\delta(\Pi, r) > \delta(\Pi', r)$$

Since $(\forall R), (\forall L), (m)$ and (w) rules do not change the space weight δ, without loss of generality we can assume that Π ends as follows:

$$\frac{\dfrac{\dfrac{\Pi_1 \triangleright \Gamma_1, \mathtt{x} : \sigma \vdash \mathtt{M} : A}{\Gamma_1 \vdash \lambda \mathtt{x.M} : \sigma \multimap A}\ (\multimap I) \qquad \Pi_2 \triangleright \Gamma_2 \vdash \mathtt{N} : \sigma}{\Gamma_1, \Gamma_2 \vdash (\lambda \mathtt{x.M})\mathtt{N} : A}\ (\multimap E)}{!^n\Gamma_1, !^n\Gamma_2 \vdash (\lambda \mathtt{x.M})\mathtt{N} :!^n A}\ (sp)^n$$

where $\Gamma_1 \# \Gamma_2$, $\Gamma =!^n\Gamma_1, !^n\Gamma_2$, $\sigma \equiv !^n A$ for $n \geq 0$. Clearly, by definition of the space weight δ, we have $\delta(\Pi, r) = r^n(\delta(\Pi_1, r) + 1 + \delta(\Pi_2, r))$. Since $r \geq \mathtt{rk}(\Pi) \geq \mathtt{rk}(\Pi_1)$, by Lemma 14 there exists a derivation $\Pi_3 \triangleright \Gamma \vdash \mathtt{M[N/x]} : A$ such that $\delta(\Pi_3, r) \leq \delta(\Pi_1, r) + \delta(\Pi_2, r)$. Hence, we can construct Π' ending as:

$$\frac{\Pi_3 \triangleright \Gamma, \Delta \vdash \mathtt{M[N/x]} : A}{!^n\Gamma_1, !^n\Gamma_2 \vdash \mathtt{M[N/x]} :!^n A}\ (sp)^n$$

Clearly $\delta(\Pi', r) \leq r^n(\delta(\Pi_1, r) + \delta(\Pi_2, r)) < \delta(\Pi, r)$ and so the conclusion follows.

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

(2) It follows directly by the definition of the space weight δ. $\square$

It is easy to verify that (h) rules leave the space weight unchanged, since $(\mathtt{xV_1 \cdots V}_m)^{\mathcal{A}} \equiv (\mathtt{NV_1 \cdots V}_m)^{\mathcal{A}}$ if $\{\mathtt{x := N}\} \in \mathcal{A}$. Hence, a direct consequence of the above lemma is the following.

LEMMA 16. *Let* $\Pi \triangleright \mathtt{M} : \mathbf{B}$ *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* $r \geq \mathtt{rk}(\Pi)$*:*

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

PROOF. Easy, by Lemma 15. $\square$

Subject reduction does not increase the space weight.

PROPERTY 2. *Let* $\Pi \triangleright \Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to^*_{\beta\delta} \mathtt{N}$*. Then there exists* $\Pi' \triangleright \Gamma \vdash \mathtt{N} : \sigma$ *with* $\mathtt{rk}(\Pi) \geq \mathtt{rk}(\Pi')$ *such that for each* $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta(\Pi, r) \geq \delta(\Pi', r)$$

Proof. By Lemma 14 and definition of δ. $\square$

It is worth noticing that a reduction inside an `if` does not necessarily decrease the space weight δ. This is the reason why we consider a non-strict inequality in the statement of the above property.
The previous result can be extended to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ in the following way.

Property 3. *Let* $\Pi \rhd \vdash \mathtt{M} : \mathbf{B}$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *For each configuration* $\phi \in \nabla$ *such that* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$ *and* $\mathcal{C} \not\equiv \circ$ *there exist derivations* $\Sigma \rhd \vdash (\mathcal{C}[\mathtt{N}])^{\mathcal{A}} : \mathbf{B}$ *and* $\Theta \rhd \vdash (\mathtt{N})^{\mathcal{A}} : \mathbf{B}$ *such that* Θ *is a proper subderivation of* Σ *and for each* $r \geq \mathtt{rk}(\Pi)$*:*

$$\delta(\Pi, r) \geq \delta(\Sigma, r) > \delta(\Theta, r)$$

Proof. Easy. $\square$

Note that in the above property we ask for $\mathcal{C} \neq \circ$ just in order to make the second inequality strict.

4.2 Proof of PSPACE Soundness

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. In order to give an account of this space, we need to measure the increasing of the size of a term during its evaluation. The key notion for realizing this measure is that of sliced occurrence of a variable, which takes into account that in performing an `if` reduction a subterm of the subject is erased. In particular, by giving a bound on the number of sliced occurrences of variables we obtain a bound on the number of applications of the h rule in a path.

Definition 17. *The number of* sliced occurrences $n_{so}(\mathtt{x}, \mathtt{M})$ *of the variable* $\mathtt{x}$ *in* $\mathtt{M}$ *is defined as:*

$$n_{so}(\mathtt{x}, \mathtt{x}) = 1,\ \ n_{so}(\mathtt{x}, \mathtt{y}) = n_{so}(\mathtt{x}, \mathtt{0}) = n_{so}(\mathtt{x}, \mathtt{1}) = 0,$$

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

$$n_{so}(\mathtt{x},\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ ) = \max\{n_{so}(\mathtt{x}, \mathtt{M}), n_{so}(\mathtt{x}, \mathtt{N}_0), n_{so}(\mathtt{x}, \mathtt{N}_1)\}$$

A type derivation gives us some information about the number of sliced occurrences of a free variable `x` in its subject `M`.

Lemma 17. *Let* $\Pi \rhd \Gamma, \mathtt{x} :!^n A \vdash \mathtt{M} : \sigma$ *then* $n_{so}(\mathtt{x}, \mathtt{M}) \leq \mathtt{rk}(\Pi)^n$.

Proof. By induction on n.
Case $n = 0$. The conclusion follows easily by induction on Π. Base cases are trivial. In the case Π ends by $(\mathbf{B}E)$, the conclusion follows by $n_{so}(\mathtt{x}, \mathtt{M})$ definition and induction hypothesis. The other cases follow directly from the induction hypothesis remembering the side condition $\Gamma \# \Delta$ in $(\multimap E)$ case.
Case $n > 0$. By induction on Π. Base case is trivial. Let the last rule of Π be:

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

where $x\ :!^n A \in \Gamma$. By induction hypothesis $n_{so}(\mathtt{x}, \mathtt{M}') \leq \mathtt{rk}(\Sigma)^n$ and $n_{so}(\mathtt{x}, \mathtt{N}_i) \leq \mathtt{rk}(\Theta_i)^n$ for $i \in \{\mathtt{0}, \mathtt{1}\}$. By definition of rank $\mathtt{rk}(\Pi) =$

$\max\{\mathtt{rk}(\Sigma), \mathtt{rk}(\Theta_0), \mathtt{rk}(\Theta_1)\}$ and since by definition $n_{so}(\mathtt{x}, \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )$ is equal to $\max\{n_{so}(\mathtt{x}, \mathtt{M}), n_{so}(\mathtt{x}, \mathtt{N}_0), n_{so}(\mathtt{x}, \mathtt{N}_1)\}$, then the conclusion follows.
Let the last rule of Π be:

$$\frac{\Sigma \rhd \Gamma, \mathtt{x}_1 :!^{n-1}A, \dots, \mathtt{x}_m :!^{n-1}A \vdash \mathtt{N} : \mu}{\Gamma, \mathtt{x} :!^{n}A \vdash \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_m] : \mu}\ (m)$$

where $\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_m] \equiv \mathtt{M}$. By induction hypothesis $n_{so}(\mathtt{x}_i, \mathtt{N}) \leq \mathtt{rk}(\Sigma)^{n-1}$ for $1 \leq i \leq m$ and since $\mathtt{rk}(\Sigma) \leq \mathtt{rk}(\Pi)$ the conclusion follows easily. In every other case the conclusion follows directly by induction hypothesis. □

It is worth noting that the above lemma and the subject reduction property gives dynamical informations about the number of sliced occurrences of a variable.

Lemma 18. *Let $\Pi \rhd \Gamma, \mathtt{x} :!^{n}A \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \rightarrow_{\beta\delta} \mathtt{N}$. Then, $n_{so}(\mathtt{x}, \mathtt{N}) \leq \mathtt{rk}(\Pi)^n$.*

Proof. Easy, by Property 2 and Lemma 17. □

The lemma above is essential to prove the following important property.

Lemma 19. *Let $\mathtt{M} \in \mathcal{P}_d$ and $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$ then for each $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$:*

$$\#_h(\phi) \leq \#(\mathcal{A})|\mathtt{M}|^d$$

Proof. For each $[\mathtt{x}' := \mathtt{N}] \in \mathcal{A}$ the variable $\mathtt{x}'$ is a fresh copy of a variable $\mathtt{x}$ originally bound in $\mathtt{M}$. Hence, $\mathtt{M}$ contains a subterm $(\lambda\mathtt{x}.\mathtt{P})\mathtt{Q}$ and there exists a derivation Π such that $\Pi \rhd \mathtt{x} :!^n A \vdash \mathtt{P} : B$.
By Lemma 18 for every $\mathtt{P}'$ such that $\mathtt{P} \rightarrow^*_{\beta\delta} \mathtt{P}'$ we have $n_{so}(\mathtt{x}, \mathtt{P}') \leq \mathtt{rk}(\Pi)^n$. So, in particular the number of applications of h rules on the variable $\mathtt{x}'$ is bounded by $\mathtt{rk}(\Pi)^n$. Since $|\mathtt{M}| \geq \mathtt{rk}(\Pi)$ and $d \geq n$, the conclusion follows. □

The following lemma relates the space weight with both the size of the term and the degree of the derivation.

Lemma 20. *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 dimensions of all the components of a machine configuration, namely the term, the m-context and the **B**-context.

Lemma 21. *Let $\mathtt{M} \in \mathcal{P}_d$ and $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. Then for each $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$:*

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

Proof.

(1) By Lemma 13.1, Lemma 16 and Lemma 20.3.

$$|\mathcal{A}| \leq \#_\beta(\phi)(|\mathtt{M}|+1) \leq \delta(\Pi, \mathtt{rk}(\Pi))(|\mathtt{M}|+1) \leq |\mathtt{M}|^{\mathtt{d}+1}(|\mathtt{M}|+1) \leq 2|\mathtt{M}|^{d+2}$$

(2) By Lemma 13.2, Lemma 19, Lemma 12.1, Lemma 16 and Lemma 20.3:

$$|\mathtt{N}| \leq (\#_h(\phi)+1)|\mathtt{M}| \leq (\#(\mathcal{A})|\mathtt{M}|^d+1)|\mathtt{M}| \leq \#_\beta(\phi)|\mathtt{M}|^{d+1} + |\mathtt{M}| \leq 2|\mathtt{M}|^{2d+2}$$

(3) By Lemma 13.3, Lemma 12.2, the previous point of this lemma, Lemma 16 and Lemma 20.3:

$$\begin{aligned}|\mathcal{C}| &\leq \#_{\mathtt{if}}(\phi)(\max\{|\mathtt{N}| \mid \psi \succ \mathcal{C}', \mathcal{A}' \models \mathtt{N} \Downarrow \mathtt{b}'' \in \mathtt{path}(\phi)\}) \\ &\leq \#(\mathcal{C})2|\mathtt{M}|^{2d+2} \leq |\mathtt{M}|^{d+1}2|\mathtt{M}|^{2d+2} \leq 2|\mathtt{M}|^{3d+3} \quad \square\end{aligned}$$

The PSPACE soundness follows immediately from the definition of $\mathtt{space}(\nabla)$, for a machine evaluation ∇, and from the previous lemma.

Theorem 4 Polynomial Space Soundness.
Let $\mathtt{M} \in \mathcal{P}_d$. *Then:*

$$\mathtt{space}(\mathtt{M}) \leq 6|\mathtt{M}|^{3d+3}$$

Proof. By definition of $\mathtt{space}(\mathtt{M})$ and Lemma 21. $\square$

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.

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

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 $\lambda\mathtt{x}.\mathtt{xMN}$ and $\mathtt{let\ z\ be\ x, y\ in\ N}$ stands for $\mathtt{z}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{N})$. Note that, since $\mathrm{STA}_{\mathbf{B}}$ 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.

$\mathbf{B}$-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 18. *Let $f : \mathbb{I}_1 \times \cdots \times \mathbb{I}_n \to \mathbb{O}$ be a total function, let $\mathbf{O}, \mathbf{I}_1, \ldots, \mathbf{I}_n \in \mathcal{T}_{\mathbf{B}}$ and let elements $o \in \mathbb{O}$ and $i_j \in \mathbb{I}_j$, for $0 \leq j \leq n$, be encoded by terms $\underline{\mathtt{o}}$ 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 $\vdash \underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n} : \mathbf{O}$ and:*

$$f i_1 \cdots i_n = o \iff \underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n} =_\beta \underline{\mathtt{o}}$$

(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 $\underline{\mathtt{f}}\underline{\mathtt{i}_1} \ldots \underline{\mathtt{i}_n} \in \mathcal{P}$ 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 22 [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

It is worth noting that due 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:

$\langle \mathtt{1},\mathtt{0}\rangle$	`A`	$\langle \mathtt{1},\mathtt{1}\rangle$	`R`	$\langle \mathtt{0},\mathtt{1}\rangle$	$\wedge$	$\langle \mathtt{0},\mathtt{0}\rangle$	$\vee$

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:

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

where $\mathtt{cb}_0^l \circ \ldots \circ \mathtt{cb}_n^l$ and $\mathtt{cb}_0^r \circ \ldots \circ \mathtt{cb}_n^r$ are respectively the left and right hand side words on the ATM tape, `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 `0` and `1`, the scanned symbol is the first symbol in the right part and final states are divided in accepting and rejecting.
Each term representing a configuration can be typed by indexed types (for every $i \geq 1$) as:

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

We need some terms defining operations on ATM. In particular, the term $\mathtt{Init} \doteq \lambda \mathtt{t}.\lambda \mathtt{c}.\langle \lambda \mathtt{z}.\mathtt{z}, \lambda \mathtt{z}.\mathtt{t}(\mathtt{c}\mathbf{0})\mathtt{z}, \langle \underline{\mathtt{Q}_0}, \underline{\mathtt{k}_0}\rangle\rangle$ 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 $\mathtt{Q}_0 \equiv \langle \mathtt{q}_0, \ldots, \mathtt{q}_n\rangle$ 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 $\mathtt{Init} : \mathbf{S}_i \multimap \mathbf{ATM}_i$ for every $i \geq 1$.
An ATM transition relation δ can be considered as the union of the transition functions $\delta_1, \ldots, \delta_n$ of its components. So, we need to show that transition functions

are definable. We decompose an ATM transition step in two stages. In the first stage, the ATM configuration is decomposed to extract the information needed by the transition relation. In the second one, the previously obtained information are combined, depending on the considered transition function δ_j, in order to build the new ATM configuration. The term performing the decomposition stage is:

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

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

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

The transition combination stage is performed by the term

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

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

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

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

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

The term that takes a configuration and return its kind is:

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

which is typable as $\vdash \mathtt{Kind} : \mathbf{ATM}_\mathtt{i} \multimap \mathbf{B}^2$. Finally the term

$$\mathtt{Ext} \doteq \lambda \mathtt{x}.\mathtt{let}\ (\mathtt{Kind}\ \mathtt{x})\ \mathtt{be}\ \mathtt{l},\mathtt{r}\ \mathtt{in}\ \mathtt{r}$$

typable as $\vdash \mathtt{Ext} : \mathbf{ATM}_\mathtt{i} \multimap \mathbf{B}$, returns 0 or 1 according to the fact that a given configuration is either accepting or rejecting.

Evaluation function

Given an ATM $\mathcal{M}$ working in polynomial time we define a recursive evaluation procedure $\mathtt{eval}_\mathcal{M}$ that takes a string s and returns 0 or 1 if the initial configuration (with the tape filled with 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 $\mathtt{Tr}^1_{\mathcal{M}}$ and $\mathtt{Tr}^2_{\mathcal{M}}$ 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 as

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

This can be defined by the term

$$\alpha(\mathtt{M}_0,\mathtt{M}_1,\mathtt{M}_2)\doteq\mathtt{let}\ \mathtt{M}_0\ \mathtt{be}\ \mathtt{a}_1,\mathtt{a}_2\ \mathtt{in}\ \ \mathtt{if}\ \mathtt{a}_1\ \mathtt{then}\ (\ \mathtt{if}\ \mathtt{a}_2\ \mathtt{then}\ \langle\mathtt{a}_1,\\ \pi_2(\mathtt{M}_1)\ \mathtt{or}\ \pi_2(\mathtt{M}_2)\rangle\ \mathtt{else}\ \langle\mathtt{a}_1,\pi_2(\mathtt{M}_1)\ \mathtt{and}\ \pi_2(\mathtt{M}_2)\rangle)\ \mathtt{else}\ \langle\mathtt{a}_1,\mathtt{a}_2\rangle$$

It is worth noting 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

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

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

$$\begin{array}{l}\vdash\mathtt{Base}:\mathbf{ATM}_i\multimap\mathbf{B}^2\\ \vdash\mathtt{Step}_{\mathcal{M}}:(\mathbf{ATM}_i\multimap\mathbf{B}^2)\multimap\mathbf{ATM}_i\multimap\mathbf{B}^2\end{array}$$

Let P be a polynomial definable by a term $\mathtt{P}$ 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:

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

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. Hence the term `Ext` can be applied with the intended meaning.

LEMMA 23. *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. $\mathcal{D}(s)=b\iff\mathtt{eval}_{\mathcal{M}}\mathtt{s}\Downarrow\mathtt{b}$ $\square$

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

THEOREM 5 POLYNOMIAL SPACE COMPLETENESS. *Every decision problem* $\mathcal{D}\in\mathrm{PSPACE}$ *is* $\mathbf{B}$*-programmable in* $\mathrm{STA}_{\mathbf{B}}$.

6. CONCLUSION

In this paper we have designed $\mathrm{STA}_{\mathbf{B}}$, a language correct and complete with respect 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 $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ 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.

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 $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$ 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. An interesting fact in the completeness proof is that the modal part of the $\mathrm{STA}_{\mathbf{B}}$ system is only involved in the polynomial iteration, while the ATM behaviour (i.e. the α function) can be defined in the modal free fragment of the system. On the basis of these facts, we think that our tools could be fruitfully used in order to revisit some classical complexity results relating time and space [Stockmeyer 1976].

Starting from the type system $\mathrm{STA}_{\mathbf{B}}$ presented in this paper, one would wonder to exploit the proofs-as-programs correspondence in the design of a purely logical characterization of the class PSPACE. In particular, one would understand 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, it looks not so easy to understand how to transfer the complexity bound from the term evaluation to the cut-elimination in a logic. Moreover, boolean constants are redundant and the $\mathrm{STA}_{\mathbf{B}}$ rule $(\mathbf{B}E)$ has no direct correspondent in sequent calculus. All these difficulties suggest that exploring this direction could be a true test for the light logics principles.

REFERENCES

Abiteboul, S. and Vianu, V. 1989. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, Washington, D.C., 71–79.

Asperti, A. and Roversi, L. 2002. Intuitionistic light affine logic. *ACM Transactions on Computational Logic 3(1)*, 137–175.

Baillot, P. and Terui, K. 2004. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS 2004. IEEE Computer Society.* 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.

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 *TLCA'05*. 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 *ESOP*, A. D. Gordon, Ed. Lecture Notes in Computer Science, vol. 6012. Springer, 205–225.

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 *35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL 2008, San Francisco, January 10-12, 2008, Proceedings*. 121–131.

GABOARDI, M., MARION, J.-Y., AND RONCHI DELLA ROCCA, S. 2008b. Soft linear logic and polynomial complexity classes. In *Proceedings of the Second Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2007)*. Electronic Notes in Theoretical Computer Science, vol. 205. Elsevier, 67–87.

GABOARDI, M. AND RONCHI DELLA ROCCA, S. 2007. A soft type assignment system for λ-calculus. In *Computer Science Logic, 21st International Workshop, CSL 07, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*. Lecture Notes in Computer Science, 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.

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

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. *Theor. Comput. Sci. 100,* 1, 45–66.

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. *J. Funct. Program. 11,* 1, 55–94.

KAHN, G. 1987. Natural semantics. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS)*. 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 *Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*. Lecture Notes in Computer Science, vol. 664. Springer, 274–288.

LEIVANT, D. AND MARION, J.-Y. 1994. Ramified recurrence and computational complexity II: Substitution and poly-space. In *CSL*. LNCS, vol. 933. Springer, 486–500.

LEIVANT, D. AND MARION, J.-Y. 1997. Predicative functional recurrence and poly-space. In *TAPSOFT '97: Theory and Practice of Software Development*. Lecture Notes in Computer Science, vol. 1214. Springer-Verlag, 369–380.

Maurel, F. 2003. Nondeterministic light logics and NP-time. In *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings*, M. Hofmann, Ed. Lecture Notes in Computer Science, vol. 2701. Springer, 241–255.

Oitavem, I. 2001. Implicit characterizations of pspace. In *Proof Theory in Computer Science, International Seminar, PTCS 2001, Dagstuhl Castle, Germany, October 7-12, 2001, Proceedings*. Lecture Notes in Computer Science, vol. 2183. Springer, 170–190.

Oitavem, I. 2008. Characterizing pspace with pointers. *Math. Log. Q. 54,* 3, 323–329.

Plotkin, G. D. 2004. A structural approach to operational semantics. *J. Log. Algebr. Program. 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 *CSL*, Z. Ésik, Ed. Lecture Notes in Computer Science, vol. 4207. Springer, 606–621.

Schöpp, U. 2007. Stratified bounded affine logic for logarithmic space. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Washington, DC, USA, 411–420.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theor. Comput. Sci. 3,* 1, 1–22.

Terui, K. 2000. Linear logical characterization of polyspace functions (extended abstract). Unpublished.

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