A Logical Account of PSPACE

Marco Gaboardi, Jean-Yves Marion, Simona Ronchi Della Rocca

A Logical account of PSPACE

Marco Gaboardi
Simona Ronchi Della Rocca
Dipartimento di Informatica, Università degli Studi di Torino - Corso Svizzera 185, 10149 Torino, Italy
gaboardi,ronchi@di.unito.it

Jean-Yves Marion
Nancy-University, ENSMN-INPL, Loria
B.P. 239, 54506 Vandoeuvre-lès-Nancy, France
Jean-Yves.Marion@loria.fr

Abstract

We propose a characterization of PSPACE by means of a type assignment for an extension of lambda calculus with a conditional construction. The type assignment $\mathrm{STA}_{\mathbf{B}}$ is an extension of STA, a type assignment for lambda-calculus inspired by Lafont's Soft Linear Logic.
We extend STA by means of a ground type and terms for booleans. The key point is that the elimination rule for booleans is managed in an additive way. Thus, we are able to program polynomial time Alternating Turing Machines. Conversely, we introduce a call-by-name evaluation machine in order to compute programs in polynomial space. As far as we know, this is the first characterization of PSPACE which is based on lambda calculus and light logics.

1. Introduction

The argument of this paper fits in the so called Implicit Computational Complexity topic, in particular on 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. If the considered resource is the time, the natural choice is to use as types formulae of the light logics, which characterize some classes of time complexity. Light Linear Logic (LLL) of Girard [Gir98], and Soft Linear Logic (SLL) of Lafont [Laf04] characterize polynomial time, while Elementary Linear Logic (EAL) characterizes elementary time. The characterization is based on the fact that cut-elimination 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 all these logics are also complete with respect to the related complexity class.
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 complexity bound. Namely every well typed term β-reduces to normal form in a number of steps which 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 [BT04] and [GR07], based respectively on LAL (an affine variant of LLL designed by Asperti and Roversi [AR02]) and on SLL, and an example of an elementary type assignment system is in [CDLRDR05].
Here we use the same approach for studying space complexity, in particular we build a type system for a λ-calculus like language, in such a way that well typed terms are correct and complete for PSPACE. More precisely, every well typed program reduces in polynomial space and all decision functions computable in polynomial space are computed by well typed programs. There is no previous logical characterization of PSPACE from which we can start. But we will use the fact that polynomial space computations coincide with polynomial time alternating Turing machine computations (APTIME) [Sav70, CKS81]. In particular

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

So we will start from the type assignment STA for λ-calculus presented in [GR07], which 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 a type assignment system ($\mathrm{STA}_{\mathbf{B}}$), where the types are STA types plus a type $\mathbf{B}$ for booleans, and the language $\Lambda_{\mathcal{B}}$ is an extension of λ-calculus with two boolean constants and a conditional constructor. 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 the if-rule above, 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, a call-by-value evaluation can construct exponential size term. So we define a call-by-name evaluation machine, inspired by Krivine's machine [Kri07] for λ-calculus, where substitutions are made only on head variables. This machine is equipped with a memory device thanks to which the space used is easily determined, as the dimension of the maximal machine configuration. Then we prove that, if the machine takes a program (i.e., a closed term well typed with a ground type) as input, then each configuration is bounded in a polynomial way 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 term of $\mathrm{STA}_{\mathbf{B}}$. The simulation relies on a higher order representation of a parameter substitution recurrence schema which was used in [LM94].

$\mathrm{STA}_{\mathbf{B}}$ is the first characterization of PSPACE through a type assignment system. A proposal for a similar characterization has been made by Terui [Ter00], but the work has never been completed.

There are previous implicit characterizations of polynomial space computations. The characterizations in [LM94, LM97] and [Oit01] 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 [Var82, AV89]. The reader may consult the recent book [GKL$^+$07]. Finally there are some algebraic characterizations like the one [Goe92] or [Jon01] but which are, in essence, over finite domains.

An example of a characterization of a complexity space class through a light logic is in [Sch07] where a logical system characterizing logarithmic space computations is defined, the Stratified Bounded Affine Logic (SBAL). Logarithmic space soundness is proved by considering only proofs of certain sequents to represent the functions computable in logarithmic space.
Our characterization is strongly based on the additive rule $(\mathbf{B}E)$. A similar tool has been used by Maurel's Non Deterministic Light Logic (nLLL) [Mau03] in order to characterize non deterministic polynomial time. More precisely nLLL introduces an explicit sum rule to deal with non deterministic computation.
A different approach but also based on linear logic is to characterize circuit complexity classes by means of Booleans proof nets [Ter04, MR07] but this does not relate to the task of designing programming language with an intrinsically polynomial computational bound.

Outline of the paper In Section 2 the system $\mathrm{STA}_{\mathbf{B}}$ is introduced and the proofs of subject reduction and strong normalization properties are given. In Section 3 the operational semantics of $\mathrm{STA}_{\mathbf{B}}$ program is defined, through an abstract evaluation machine. In Section 4 we show that $\mathrm{STA}_{\mathbf{B}}$ programs can be executed in polynomial space. In Section 5 the completeness for PSPACE is proved. In Section 6 some complementary argument are considered. The Appendix contains the most technical proofs.

2. Soft Type Assignment system with Booleans

In this section the type assignment $\mathrm{STA}_{\mathbf{B}}$ is presented, and its properties are proved. $\mathrm{STA}_{\mathbf{B}}$ is an extension of the type system STA for λ-calculus introduced in [GR07], which assigns to terms of the λ-calculus a proper subset of formulae of Lafont's Soft Linear Logic [Laf04]. STA has been proved to be correct and complete for polynomial time computations.
$\mathrm{STA}_{\mathbf{B}}$ is obtained from STA by extending both the calculus and the set of types. The calculus is the λ-calculus extended by boolean constants $\mathtt{0}, \mathtt{1}$ and an `if` constructor, types are the types of STA plus a constant type $\mathbf{B}$ for booleans.

Definition 1.

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 set* $\mathcal{T}$ *of* $\mathbf{B}$ 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* $\mathbf{B}$ *is the only* ground *type.*

3. *A context is a set of assumptions of the shape* $\mathtt{x} : \sigma$*, where all variables are different. We use* Γ, Δ *to denote contexts.* $\mathrm{dom}(\Gamma) = \{\mathtt{x} \mid \exists \mathtt{x} : \sigma \in \Gamma\}$ *and* $\Gamma \# \Delta$ *means* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$*.*

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

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

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

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

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

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

Table 1. The Soft Type Assignment system with Booleans

4. $\mathrm{STA}_{\mathbf{B}}$ *proves judgments of the shape* $\Gamma \vdash \mathtt{M} : \sigma$ *where* Γ *is a context,* $\mathtt{M}$ *is a term, and* σ *is a* $\mathbf{B}$ *type. The rules are given in Table 1.*
5. *Derivations are denoted by* $\nabla, \Diamond, \Box$. $(\nabla)\ \Gamma \vdash \mathtt{M} : \sigma$ *denotes a derivation* ∇ *with conclusion* $\Gamma \vdash \mathtt{M} : \sigma$*. We let* $\vdash \mathtt{M} : \sigma$ *abbreviate* $\emptyset \vdash \mathtt{M} : \sigma$*.*

Note that while all rules in STA have a multiplicative treatment of contexts, the rule $(\mathbf{B}E)$ of $\mathrm{STA}_{\mathbf{B}}$ is additive and so contraction is free.

Notation 1. *Terms are ranged over by* $\mathtt{M}, \mathtt{N}, \mathtt{V}, \mathtt{P}$*. As usual terms are considered up to* α*-equivalence, namely a bound variable can be renamed provided no free variable is captured. Moreover* $\mathtt{M}[\mathtt{N}/\mathtt{x}]$ *denotes the capture-free substitution of all free occurrences of* $\mathtt{x}$ *in* $\mathtt{M}$ *by* $\mathtt{N}$. $\mathrm{FV}(\mathtt{M})$ *denotes the set of free variables of* $\mathtt{M}$*,* $n_o(\mathtt{x}, \mathtt{M})$ *the number of free occurrences of the variable* $\mathtt{x}$ *in* $\mathtt{M}$*.*
Type variables are ranged over by α, β*, linear types by* A, B, C*, and types by* σ, τ, μ*.* $\equiv$ *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.* $\sigma[A/\alpha]$ *denotes the capture free substitution in* σ *of all occurrences of the type variable* α *by the linear type* A*.* $\mathrm{FTV}(\Gamma)$ *denotes the set of free type variables occurring in the assumptions of the context* Γ*.*
We stress that each type is of the shape $!^n \forall \vec{\alpha}.A$ *where* $\forall \vec{\alpha}.A$ *is an abbreviation for* $\forall \alpha_1 \ldots . \forall \alpha_m.A$*, and* $!^n \sigma$ *is an abbreviation for* $! \ldots ! \sigma$ n*-times. In particular* $!^0 \sigma \equiv \sigma$*.*

We have the following standard properties for a natural deduction system.

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* $\Delta \vdash \mathtt{M} : \sigma$*.*

The functional behaviour of $\Lambda_{\mathcal{B}}$ is described in the next definition.

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

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

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

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

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

In what follows, we will need to talk about proofs modulo commutations of rules.

Definition 3. *Let* Π *and* Π' *be two derivations in* $\mathrm{STA}_{\mathbf{B}}$*, proving the same conclusion:* $\Pi \rightsquigarrow \Pi'$ *denotes the fact that* Π' *is obtained from* Π *by commuting or deleting some rules.*

The Generation Lemma connects the shape of a term with its possible typings, and will be useful in the sequel.

Lemma 2 (Generation lemma)**.**

1. $(\nabla)\ \Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \forall \alpha.A$ *implies there is* ∇' *such that* $\nabla \rightsquigarrow \nabla'$ *where the last rule of* ∇' *is* $(\forall I)$.
2. $(\nabla)\ \Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A$ *implies there is* ∇' *such that* $\nabla \rightsquigarrow \nabla'$*, whose last rule is* $(\multimap I)$.
3. $(\nabla)\ \Gamma \vdash \mathtt{M} :!\sigma$ *implies there is* ∇' *such that* $\nabla \rightsquigarrow \nabla'$ *where* ∇' *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).
4. $(\nabla)\ !\Gamma \vdash \mathtt{M} :!\sigma$ *implies there is* ∇' *such that* $\nabla \rightsquigarrow \nabla'$*, whose last rule is* (sp).

The substitution lemma will be the key lemma to show that $\mathrm{STA}_{\mathbf{B}}$ enjoys the subject reduction property.

Lemma 3 (Substitution lemma)**.** *Let* $(\nabla)\ \Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ *and* $(\Diamond)\ \Delta \vdash \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$*. Then there exists* $(\Box)\ \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$.

Proof. Since the proof is quite involved, we postpone it to Appendix A.1. ☐

We can finally prove the main property of this section.

Lemma 4 (Subject Reduction)**.** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \rightarrow_{\beta\delta} \mathtt{N}$*. Then* $\Gamma \vdash \mathtt{N} : \sigma$.

Proof. The case of a $\rightarrow_{\delta}$ reduction is easy, the one of $\rightarrow_{\beta}$ reduction follows by Lemma 3. ☐

By strong normalization of STA we have the following.

Lemma 5 (Strong Normalization)**.** *Let* $\Gamma \vdash \mathtt{M} : \sigma$ *then* $\mathtt{M}$ *is strongly normalizing with respect to the reduction relation* $\rightarrow_{\beta\delta}$.

Nevertheless 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 in it.

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

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

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

3. Structural Operational Semantics

In this section the operational semantics of terms of $\Lambda_{\mathcal{B}}$ will be given, through an evaluation machine, defined in SOS style, performing the evaluation according to the leftmost outermost strategy. The machine, if restricted to λ-calculus, is quite similar to the Krivine machine[Kri07], 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 evaluation machine is related to the type assignment system $\mathrm{STA}_{\mathbf{B}}$ in the sense that, when it starts on an empty memory, all the programs (closed terms of ground type) can be evaluated.

Definition 4. *The set* $\mathcal{P}$ *of* $\mathrm{STA}_{\mathbf{B}}$ programs *is the set of closed terms typable by the ground type. i.e.* $\mathcal{P} = \{\mathtt{M} \mid \vdash \mathtt{M} : \mathbf{B}\}$.

It is easy to check that a $\mathrm{STA}_{\mathbf{B}}$ term is of the following shape:

$$\mathtt{M} \equiv \lambda \mathtt{x}_1 ... \mathtt{x}_n.\zeta \mathtt{V}_1 \cdots \mathtt{V}_m$$

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 $\mathtt{if}\ \mathtt{P}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1$.

In particular, if a term is a program, then its shape is as before, but with the condition that the number n of initial abstractions is equal to 0. Moreover if ζ is a boolean $\mathtt{b}$ then the number m of arguments is equal to 0. We will use this characterization of programs to design the evaluation machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$.

$\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ uses two memory devices, the m-context and the $\mathbf{B}$-context, that memorize respectively the assignments to variables and the control.

Definition 5.

1. *An* m-context $\mathcal{A}$ *is a sequence of variable assignments of the shape* $\mathtt{x}_i := \mathtt{M}_i$ *where all variables* $\mathtt{x}_i$ *are distinct. The set of m-contexts is denoted by* $\mathrm{Ctx}_{\mathrm{m}}$.
2. *The* cardinality *of an m-context* $\mathcal{A}$*, denoted by* $\#(\mathcal{A})$*, is the number of variable assignments in* $\mathcal{A}$.
3. *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* $|\mathtt{M}| + 1$*, and* $|\mathtt{M}|$ *is the number of symbols of* $\mathtt{M}$.
4. *Let* $\circ$ *be a distinguished symbol. The set* $\mathrm{Ctx}_{\mathbf{B}}$ *of* $\mathbf{B}$-contexts *is defined by the following grammar:*
$$\mathcal{C}[\circ] ::= \circ \mid (\ \mathtt{if}\ \mathcal{C}[\circ]\ \mathtt{then}\ \mathtt{M}\ \mathtt{else}\ \mathtt{N}\ )\mathtt{V}_1 \cdots \mathtt{V}_n$$
5. *The size of a* $\mathbf{B}$*-context* $\mathcal{C}[\circ]$ *denoted* $|\mathcal{C}[\circ]|$ *is the size of the term obtained by replacing the symbol* $\circ$ *by a variable. The cardinality of a* $\mathbf{B}$*-context* $\mathcal{C}[\circ]$*, denoted by* $\#(\mathcal{C}[\circ])$*, is the number of nested* $\mathbf{B}$*-contexts in it.*

Notation 2. ε *denotes the empty m-context and* $\mathcal{A}_1 @ \mathcal{A}_2$ *denotes the concatenation of the m-contexts* $\mathcal{A}_1$ *and* $\mathcal{A}_2$. $[\mathtt{x} := \mathtt{M}] \in \mathcal{A}$ *denotes the fact that* $\mathtt{x} := \mathtt{M}$ *is in the m-context* $\mathcal{A}$. $\mathrm{FV}(\mathcal{A}) = \bigcup_{[\mathtt{x}:=\mathtt{M}] \in \mathcal{A}} \mathrm{FV}(\mathtt{M})$.
As usual $\mathcal{C}[\mathtt{M}]$ *denotes the term obtained by filling the hole* $[\circ]$ *in* $\mathcal{C}[\circ]$ *by* $\mathtt{M}$*. In general we omit the hole* $[\circ]$ *and we range over* $\mathbf{B}$*-contexts by* $\mathcal{C}$. $\mathrm{FV}(\mathcal{C}) = \mathrm{FV}(\mathcal{C}[\mathtt{M}])$ *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 6. *Let* $\mathcal{A} = [\mathtt{x}_1 := \mathtt{N}_1, \ldots, \mathtt{x}_n : \mathtt{N}_n]$ *be an m-context. Then* $(\ )^{\mathcal{A}} : \Lambda_{\mathcal{B}} \rightarrow \Lambda_{\mathcal{B}}$ *is the map associating to each term* $\mathtt{M}$ *the term* $\mathtt{M}[\mathtt{N}_n/\mathtt{x}_n][\mathtt{N}_{n-1}/\mathtt{x}_{n-1}] \cdots [\mathtt{N}_1/\mathtt{x}_1]$.

$\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ is defined in Table 2. Some comments are in order. The rules will be commented bottom-up, which is the natural direction of the evaluation flow.

Rule (Ax) is obvious. Rule (β) 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. Rule (h) replaces the head occurrence of the head variable by the term associated with it in the m-context. Rules ($\mathtt{if}\ 0$) and ($\mathtt{if}\ 1$) perform the δ reductions. Here the evaluation naturally erases part of the subject, but the erased information is

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

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

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

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

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

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

Table 2. The Abstract Machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$

stored in the $\mathbf{B}$-context. In order to state formally the behaviour of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ we need a further definition.

Definition 7.

1. *The* evaluation relation $\Downarrow \subseteq \mathrm{Ctx}_{\mathbf{B}} \times \mathrm{Ctx}_{\mathrm{m}} \times \Lambda_{\mathcal{B}} \times \mathcal{B}$ *is the effective relation inductively defined by the rules of* $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$*. If* $\mathtt{M}$ *is a program, and if there is a boolean* b *such that* $[\circ], \varepsilon \models \mathtt{M} \Downarrow b$ *then we say that* $\mathtt{M}$ evaluates*, and we write* $\mathtt{M} \Downarrow$*.* $\models \mathtt{M} \Downarrow b$ *is a short for* $[\circ], \varepsilon \models \mathtt{M} \Downarrow b$*.*
2. *Derivation trees are called* computations *in the abstract machine and are denoted by* Π, Σ*.* $\Pi :: \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *denotes a computation with conclusion* $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$*.*
3. *Given a computation* Π *each node of* Π*, which is of the shape* $\mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *is a* configuration*. Configurations are ranged over by* ϕ, ψ*.* $\phi \rhd \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *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* Π*, the* path *to reach a configuration* ϕ *denoted* $\mathtt{path}_{\Pi}(\phi)$ *is the sequence of configurations between the conclusion of* Π *and* ϕ*. In general we will write* $\mathtt{path}(\phi)$ *when* Π *is clear from the context.*

We are interested in the behaviour of the machine when applied to programs. By an analysis of the rules of Table 2, and from the previous comments, it is easy to verify the following.

Lemma 6.

1. *Let* $\mathtt{M} \in \mathcal{P}$*,* $\Pi :: \models \mathtt{M} \Downarrow \mathtt{b}$ *and* $\mathcal{C},\mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \Pi$*. Then* $(\mathcal{C}[\mathtt{N}])^{\mathcal{A}}, (\mathtt{N})^{\mathcal{A}} \in \mathcal{P}$*.*
2. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\Pi :: \models \mathtt{M} \Downarrow \mathtt{b}$*. For each* $\mathcal{C},\mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \Pi$

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

Note that, in the previous lemma, the m-context and the $\mathbf{B}$-context are essential in proving the desired properties. In fact the $\mathbf{B}$-context recovers the part of the term necessary to complete δ-reductions, while the m-context completes β-reductions that have been performed only partially by the machine.

Example 2. *In Table 5 we present an example of* $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ *computation on the same term of Example 1.*

Note that by Definition 7.1 a term $\mathtt{M}$ evaluates only if it is a program and there exists $\mathtt{b}$ such that $\models \mathtt{M} \Downarrow \mathtt{b}$. We stress here, that the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ is complete with respect to programs, in the sense that all the programs can be evaluated.

Theorem 1.

$$\mathtt{M} \in \mathcal{P} \textit{ implies } \mathtt{M} \Downarrow$$

Proof. By induction on the reduction to normal form using Lemma 5. □

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

In Table 3 we depict a small step version of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. The rules are similar to the rules in Table 2 but the use of a garbage collector procedure described in Table 4 which is needed in order to maintain the desired complexity property. In fact the small step machine can be easily shown equivalent to the big step one.
The small step machine explicit the evaluation order clearifying that every configuration depends uniquely on the previous one (thanks to the $\mathbf{B}$-context). So the space necessary to evaluate a program turns out to be the maximum space used by one of its configurations.
Nevertheless, the big step machine has the advantage of being more abstract and this make it easy to prove the complexity properties. In fact, the garbage collector procedure make more difficult the proofs of such properties for the small step machine. For this reason in what follows we will work on the big step machine.

3.2 Space Measures

We can now define the space effectively used to evaluate a term. The remarks in the previous section allows us to consider the following definition.

Definition 8. *Let* $\phi \rhd \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *be a configuration then its* size *denoted* $|\phi|$ *is the sum* $|\mathcal{C}| + |\mathcal{A}| + |\mathtt{M}|$*. Let* $\Pi :: \mathcal{C},\mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *be a computation, then its* space occupation *denoted* $\mathtt{space}(\Pi)$ *is the maximal size of a configuration in* Π*.*

In particular since there is a one-to-one correspondence between a program $\mathtt{M}$ and its computation $\Pi :: [\circ], \varepsilon \models \mathtt{M} \Downarrow \mathtt{b}$, we will usually write $\mathtt{space}(\mathtt{M})$ in place of $\mathtt{space}(\Pi)$. In order to have polynomial space soundness we will show that for each $\mathtt{M} \in \mathcal{P}$ there exists a polynomial $P(X)$ such that $\mathtt{space}(\mathtt{M}) \le P(|\mathtt{M}|)$. The result will be proved in next section.

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

$$\frac{}{\langle \mathcal{C}, \mathcal{A}_1@[\mathtt{x} := \mathtt{N}]@\mathcal{A}_2 \succ \mathtt{x}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}, \mathcal{A}_1@[\mathtt{x} := \mathtt{N}]@\mathcal{A}_2 \succ \mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \rangle} \ (h_0)$$

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

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

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

Table 3. The small step machine $\mathrm{k}_{\mathcal{B}}^{\mathcal{C}}$

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

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

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

Table 4. The garbage collector procedure.

Example 3. *By returning to the computation of Example 2 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 $\to_\delta$ step* $(\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1\ )^{\mathcal{A}_3} \to_\delta (\mathtt{x}_1)^{\mathcal{A}_3}$ *noting that* $(\mathtt{x}_1)^{\mathcal{A}_3} \equiv (\mathtt{x}_1)^{\mathcal{A}_2}$. *In fact this can be generalized, so in this sense we don't need neither mechanism for backtracking nor the memorization of parts of the computation tree.*

In what follows we will introduce some relations between the size of the contexts and the behaviour of the machine, which will be useful later.

Definition 9. *Let Π be a computation and $\phi \in \Pi$ a configuration.*

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

The cardinality of the contexts is a measure of the number of some rules performed by the machine.

Lemma 7. *Let* $\Pi ::\models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each configuration* $\phi \rhd \mathcal{C}_i, \mathcal{A}_i \models \mathtt{P}_i \Downarrow \mathtt{b}' \in \Pi$*:*

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

The following is an important property of the machine $\mathrm{K}_{\mathcal{B}}^{\mathcal{C}}$.

Property 1. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\Pi ::\models \mathtt{M} \Downarrow \mathtt{b}$ *then for each* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \Pi$ *if* $[x_j := \mathtt{N}_j] \in \mathcal{A}_i$ *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 ϕ with minimal path from it to the root of Π, such that in its m-context $\mathcal{A}_\phi$ 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 make the m-context grow is a (β) rule we are in a situation like the following:

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

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:

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

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. $\square$

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

Lemma 8. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\Pi ::\models \mathtt{M} \Downarrow \mathtt{b}$ *then for each configuration* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \Pi$*:*

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

Proof. 1. By inspection of the rules of Table 2 it is easy to verify that m-contexts can grow only by applications of the (β) rule. So the conclusion follows by Lemma 7 and Property 1.

2. By inspection of the rules of Table 2 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 2 it is easy to verify that $\mathbf{B}$-contexts can grow only by applications of $(\mathtt{if}\ 0)$ and $(\mathtt{if}\ 1)$ rules. So the conclusion follows directly by Lemma 7. ☐

4. PSPACE Soundness

In this section we will show that $\mathrm{STA}_\mathbf{B}$ is correct for polynomial space computation, namely 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. The degree of a derivation counts the maximum nesting of applications of the rule (sp) in it. So 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 do not distinguish 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 10.

1. *The degree* $\mathtt{d}(\nabla)$ *of* ∇ *is the maximum nesting of applications of rule* (sp) *in* ∇.
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.

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

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 increment it. Moreover this result is extended to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. In the second part 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 [Laf04].

Definition 11.

1. *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 maximum rank of a rule* (m) *in* ∇. *The rank* $\mathtt{rk}(\nabla)$ *of* ∇ *is the maximum of* 1 *and* r.
2. *Let* r *be a natural number. The space weight* $\delta(\nabla, r)$ *of* ∇ *with respect to* r *is defined inductively as follows.*

   (a) *If the last applied rule is* $(Ax), (\mathbf{B}_0 I), (\mathbf{B}_1 I)$ *then* $\delta(\nabla, r) = 1$.
   (b) *If the last applied rule is* $(\multimap I)$ *with premise a derivation* $\Diamond$, *then* $\delta(\nabla, r) = \delta(\Diamond, r) + 1$.
   (c) *If the last applied rule is* (sp) *with premise a derivation* $\Diamond$, *then* $\delta(\nabla, r) = r\delta(\Diamond, r)$.
   (d) *If the last applied rule is* $(\multimap E)$ *with premises* $\Diamond$ *and* $\Box$ *then* $\delta(\nabla, r) = \delta(\Diamond, r) + \delta(\Box, r) + 1$.
   (e) *If the last applied rule is:*

$$\frac{(\Diamond)\ \Gamma \vdash \mathtt{M} : \mathbf{B} \quad (\Box_0)\ \Gamma \vdash \mathtt{N}_0 : A \quad (\Box_1)\ \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A}$$

   then $\delta(\nabla, r) = \max\{\delta(\Diamond, r), \delta(\Box_0, r), \delta(\Box_1, r)\} + 1$
   (f) *In every other case* $\delta(\nabla, r) = \delta(\Diamond, r)$ *where* $\Diamond$ *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 9 (Weighted Substitution Lemma)**.** *Let* $(\nabla)\ \Gamma, \mathtt{x} : \mu \vdash \mathtt{M} : \sigma$ *and* $(\Diamond)\ \Delta \vdash \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$ *then there exists* $(\Box)\ \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$ *such that if* $r \geq \mathtt{rk}(\nabla)$*:*

$$\delta(\Box, r) \leq \delta(\nabla, r) + \delta(\Diamond, r)$$

Proof. We postpone the proof to Appendix A.2. ☐

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 10. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\Pi ::\models \mathtt{M} \Downarrow \mathtt{b}$.

1. *Consider an occurrence in* Π *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 derivations $(\Diamond)\ \vdash ((\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} : \mathbf{B}$ *there exists a derivation* $(\Box)\ \vdash (\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@\{\mathtt{x}':=\mathtt{N}\}} : \mathbf{B}$ *such that for every* $r \geq \mathtt{rk}(\Diamond)$*:*

$$\delta(\Diamond, r) > \delta(\Box, r)$$

2. *Consider an occurrence in* Π *of an* $\mathtt{if}$ *rule as:*

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

where $\mathcal{C}' \equiv \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$. *Then, for each derivation* $(\Diamond)\ \vdash ((\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} : \mathbf{B}$ *there are derivations* $(\Box)\ \vdash (\mathtt{M})^{\mathcal{A}} : \mathbf{B}$ *and* $(\nabla)\ \vdash (\mathtt{N}_0\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} : \mathbf{B}$ *such that for every* $r \geq \mathtt{rk}(\Diamond)$*:*

$$\delta(\Diamond, r) > \delta(\Box, r) \quad \text{and} \quad \delta(\Diamond, r) > \delta(\nabla, r)$$

Proof.

1. It suffices to consider the case where $m = 0$, and to prove that, if $(\nabla)\ \Gamma \vdash (\lambda \mathtt{x}.\mathtt{M})\mathtt{N} : \sigma$, then there exists $(\nabla')\ \Gamma \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}]\sigma$ with $\mathtt{rk}(\nabla) \geq \mathtt{rk}(\nabla')$ such that for $r \geq \mathtt{rk}(\nabla)$:

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

Since $(\forall R), (\forall L), (m)$ and (w) rules don't change the δ measure we can without loss of generality assume that we are in a

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

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

Table 5. An example of computation in $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$.

situation like the following:

$$
\dfrac{\dfrac{\dfrac{(\Diamond)\ \Gamma, \mathtt{x} : \sigma \vdash \mathtt{M} : A}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap I) \qquad (\Box)\ \Delta \vdash \mathtt{N} : \sigma}{\Gamma, \Delta \vdash (\lambda \mathtt{x}.\mathtt{M})\mathtt{N} : A}\ (\multimap E)}{!^n\Gamma, !^n\Delta \vdash (\lambda \mathtt{x}.\mathtt{M})\mathtt{N} : !^n A}\ (sp)^n
$$

where $\Gamma \# \Delta$ and $n \geq 0$. Clearly we have $\delta(\nabla, r) = \delta(\Diamond, r) + 1 + \delta(\Box, r)$. By Lemma 9 we have $(\nabla')\ \Gamma \vdash \mathtt{M}[\mathtt{N}/x]\sigma$ such that $\delta(\nabla', r) \leq \delta(\Diamond, r) + \delta(\Box, r)$. Hence, the conclusion follows.

2. Easy, by definition of δ. □

Since it is easy to verify that h rules leave the space weight unchanged, a direct consequence of the above lemma is the following.

Lemma 11. *Let* $(\nabla) \vdash \mathtt{M} : \mathbf{B}$ *and* $\Pi :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \Pi$ *if* $r \geq \mathtt{rk}(\nabla)$*:*

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

Now we are ready to prove that subject reduction does not increase the space weight.

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

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

Proof. By Lemma 9 and definition of δ, noting that a reduction inside an `if` can leave δ unchanged. □

The previous result can be extended to the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ in the following way.

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

$$\delta(\nabla, r) \geq \delta(\Diamond, r) > \delta(\Box, r)$$

4.2 Proof of PSPACE Soundness

As we said 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 how the size of a term can increase during the evaluation. The key notion for doing it is that of number of the 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 occurrence we obtain a bound on the number of applications of the h rule in a path.

Definition 12. *The number of* sliced occurrences $n_{so}(\mathtt{x}, \mathtt{M})$ *of the variable* $\mathtt{x}$ *occurring free in* $\mathtt{M}$ *is defined as:*

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

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

$$n_{so}(\mathtt{x},\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ ) = \max\{n_{so}(\mathtt{x}, \mathtt{M}), n_{so}(\mathtt{x}, \mathtt{N}_0), n_{so}(\mathtt{x}, \mathtt{N}_1)\}$$

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

Lemma 12. *Let* $(\nabla)\ \Gamma, \mathtt{x} :!^n A \vdash \mathtt{M} : \sigma$ *then* $n_{so}(\mathtt{x}, \mathtt{M}) \leq \mathtt{rk}(\nabla)^n$.

Proof. By induction on n.
Case $n = 0$. The conclusion follows easily by induction on (∇). Base case is trivial. In the case (∇) ends by $(\mathbf{B}E)$ 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 (∇). Base case is trivial. Let the last rule of (∇) be:

$$\frac{(\Diamond)\ \Gamma \vdash \mathtt{M}' : \mathbf{B} \quad (\Box_0)\ \Gamma \vdash \mathtt{N}_0 : A \quad (\Box_1)\ \Gamma \vdash \mathtt{N}_1 : A}{\Gamma \vdash\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A}\ (\mathbf{B}E)$$

where $x :!^n A \in \Gamma$. By induction hypothesis $n_{so}(\mathtt{x}, \mathtt{M}') \leq \mathtt{rk}(\Diamond)^n$ and $n_{so}(\mathtt{x}, \mathtt{N}_i) \leq \mathtt{rk}(\Box_i)^n$ for $i \in \{0, 1\}$.
By definition $\mathtt{rk}(\nabla) = \max\{\mathtt{rk}(\Diamond), \mathtt{rk}(\Box_0), \mathtt{rk}(\Box_1)\}$ 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 (∇) be:

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

where $\mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_m] \equiv \mathtt{M}$. By induction hypothesis $n_{so}(\mathtt{x}_i, \mathtt{N}) \leq \mathtt{rk}(\Diamond)^{n-1}$ for $1 \leq i \leq m$. Hence in particular, $n_{so}(\mathtt{x}, \mathtt{N}) \leq m \times \mathtt{rk}(\Diamond)^{n-1}$. Now, since $m \leq \mathtt{rk}(\nabla)$ and $\mathtt{rk}(\Diamond) \leq \mathtt{rk}(\nabla)$ it follows $n_{so}(\mathtt{x}, \mathtt{N}) \leq \mathtt{rk}(\nabla) \times \mathtt{rk}(\Diamond)^{n-1} \leq rk(\nabla)^n$ and so the conclusion. In every other case the conclusion follows directly by induction hypothesis. $\Box$

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 13. *Let* $(\nabla)\ \Gamma, \mathtt{x} :!^n A \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$. *Then* $n_{so}(\mathtt{x}, \mathtt{N}) \leq \mathtt{rk}(\nabla)^n$.

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

Lemma 14. *Let* $\mathtt{M} \in \mathcal{P}_d$ *and* $\Pi\ ::\models\ \mathtt{M} \Downarrow \mathtt{b}$ *then for each* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \Pi$*:*

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

Proof. For each $[\mathtt{y} := \mathtt{N}] \in \mathcal{A}$ the variable $\mathtt{y}$ 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 $(\nabla)\ \mathtt{x} :!^n A \vdash \mathtt{P} : B$. Hence by Lemma 13 for every $\mathtt{P}'$ such that $\mathtt{P} \to^*_{\beta\delta} \mathtt{P}'$ we have $n_{so}(x, \mathtt{P}') \leq \mathtt{rk}(\nabla)^n$. In particular the number of applications of h rules on the variable $\mathtt{y}$ is bounded by $\mathtt{rk}(\nabla)^n$. Since $|\mathtt{M}| \geq \mathtt{rk}(\nabla)$ and $d \geq n$ the conclusion follows. $\Box$

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

Lemma 15. *Let* $(\nabla)\ \Gamma \vdash \mathtt{M} : \sigma$.

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

Proof. 1. By induction on (∇). 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 δ.
2. By induction on (∇). 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 δ and $\mathtt{d}$.
3. By definition of rank it is easy to verify that $\mathtt{rk}(\nabla) \leq |\mathtt{M}|$, hence by the previous two points the conclusion follows. $\Box$

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 $\mathbf{B}$-context.

Lemma 16. *Let* $\mathtt{M} \in \mathcal{P}_d$ *and* $\Pi\ ::\ \models\ \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \Pi$*:*

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 8.1, Lemma 11 and Lemma 15.3.
2. By Lemma 8.2, Lemma 14, Lemma 7.1, Lemma 11 and Lemma 15.3:
$$|\mathtt{N}| \leq (\#_h(\phi) + 1)|\mathtt{M}| \leq \#(\mathcal{A})|\mathtt{M}|^{d+1} + |\mathtt{M}| \leq 2|\mathtt{M}|^{2d+2}$$
3. By Lemma 8.3, the previous point of this lemma, Lemma 7.2, Lemma 11 and Lemma 15.3:
$$|\mathcal{C}| \leq \#(\mathcal{C})2|\mathtt{M}|^{2d+2} \leq |\mathtt{M}|^{d+1}2|\mathtt{M}|^{2d+2} \leq 2|\mathtt{M}|^{3d+3}$$

$\Box$

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

Theorem 2 (Polynomial Space Soundness)**.**
Let $\mathtt{M} \in \mathcal{P}_d$. *Then:*

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

5. PSPACE completeness

It is well known 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) [CKS81] in time polynomial in the length of the input.
In [GR07] it has been shown that polytime DTM are definable by λ-terms typable in STA. Analogously here we will show that polytime ATM are definable by programs of $\mathrm{STA}_\mathbf{B}$. We achieve such a result considering a notion of *function programmable* in $\mathrm{STA}_\mathbf{B}$. We will consider the same representation of data types as in STA, in particular data types typable through derivations with degree 0. (We will recall it briefly but we refer to [GR07] for more details.) Finally we show that for each polytime ATM $\mathcal{M}$ we can define a recursive evaluation procedure which behaves as $\mathcal{M}$.

Some syntactic sugar Let $\circ$ denote composition. In particular $\mathtt{M} \circ \mathtt{N}$ stands for $\lambda\mathtt{z}.\mathtt{M}(\mathtt{N}\mathtt{z})$ and $\mathtt{M}_1 \circ \mathtt{M}_2 \circ \cdots \circ \mathtt{M}_n$ stands for $\lambda\mathtt{z}.\mathtt{M}_1(\mathtt{M}_2(\cdots(\mathtt{M}_n\mathtt{z})))$.
Tensor product is definable as $\sigma \otimes \tau \doteq \forall\alpha.(\sigma \multimap \tau \multimap \alpha) \multimap \alpha$. In particular $\langle\mathtt{M}, \mathtt{N}\rangle$ stands for $\lambda\mathtt{x}.\mathtt{x}\mathtt{M}\mathtt{N}$ and let $\mathtt{z}$ be $\mathtt{x}, \mathtt{y}$ in $\mathtt{N}$ 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 allow the projectors: 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 $\mathtt{M}(\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y})$. n-ary tensor product can be easily defined through the binary one and we use σ^n to denote $\sigma \otimes \cdots \otimes \sigma$ n-times.

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

Lemma 17. *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 boolean are represented by terms of the shape $\lambda \mathtt{cz.cb}_0(\cdots(\mathtt{cb}_n \mathtt{z})\cdots)$ where $\mathtt{b}_i \in \{0,1\}$. 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 `len` typable as $\vdash \mathtt{len} : \mathbf{S}_i \multimap \mathbf{N}_i$ that given a string of boolean returns its length.
Note that the data types defined above can be typed in $\mathrm{STA}_{\mathbf{B}}$ by derivations with degree 0.

Programmable functions The polynomial time completeness in [GR07] relies on the notion of *λ-definability*, given in [Bar84], generalized to different kinds of data.
The same can be done here for $\mathrm{STA}_{\mathbf{B}}$, by using a generalization of λ-definability to the set of terms $\Lambda_{\mathcal{B}}$. Nevertheless this is not sufficient, since we want to show that polynomial time ATM can be defined by programs of $\mathrm{STA}_{\mathbf{B}}$. In fact we have the following definition.

Definition 13. *Let* $f : \overbrace{\mathbb{S} \times \ldots \times \mathbb{S}}^{n\text{-times}} \to \mathbb{B}$ *and let every string* $s \in \mathbb{S}$ *be representable by terms* $\mathtt{s}$*). Then,* f *is* programmable *if, there exists a term* $\mathtt{f} \in \Lambda_{\mathcal{B}}$*, such that* $\mathtt{fs}_1 \ldots \mathtt{s}_n \in \mathcal{P}$ *and:*

$$f(s_1,\ldots s_n) = b \iff \models \mathtt{fs}_1 \ldots \mathtt{s}_n \Downarrow \mathtt{b}$$

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. In fact let $\mathtt{M}$ and $\mathtt{N} \doteq$ `if M then ( if N then 0 else 1 ) else 1` and $\mathtt{M}$ or $\mathtt{N} \doteq$ `if M then 0 else ( if N then 0 else 1 )`. Then the following rules with an additive management of contexts are derived:

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

Moreover we have a term `not` defining the expected boolean function.

ATMs Configurations The encoding of Deterministic Turing Machine configuration given in [GR07] 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* (∧), *existential* (∨) . We can encode such information by tensor pairs of booleans. In particular:

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

We say that a configuration is accepting, rejecting, universal or existential depending on the kind of its state.
We can encode ATM configurations by terms of the shape:

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

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 handside words on the ATM tape, Q is a tuple of length q encoding the state and k is the tensor pair encoding the kind of state. Such terms can be typed as:

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

It is easy to adapt the term described in [GR07] dealing with TM to the case of ATM. In particular we have:

$\vdash \mathtt{Init} : \mathbf{S}_i \multimap \mathbf{ATM}_i$	Initial configuration
$\vdash \mathtt{Tr}_1, \mathtt{Tr}_2 : \mathbf{ATM}_i \multimap \mathbf{ATM}_i$	Transition functions

Moreover we have a term:

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

typable as $\vdash \mathtt{Kind} : \mathbf{ATM_i} \multimap \mathbf{B}^2$ which takes a configuration and return its kind. We also have a term

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

typable as $\vdash \mathtt{Ext} : \mathbf{ATM_i} \multimap \mathbf{B}$. Assuming that a given configuration is either accepting or rejecting Ext returns 0 or 1 respectively.

Evaluation function Given an ATM $\mathcal{M}$ working in polynomial time we define a recursive evaluation procedure $\mathtt{eval}_{\mathcal{M}}$ which 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 of degree two (at each step we consider two transitions). We need to define some auxiliary functions.

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

which acts as:

$$\alpha(\mathtt{A},\mathtt{M}_1,\mathtt{M}_2) = \mathtt{A} \qquad \alpha(\wedge,\mathtt{M}_1,\mathtt{M}_2) = \mathtt{M}_1 \wedge \mathtt{M}_2$$
$$\alpha(\mathtt{R},\mathtt{M}_1,\mathtt{M}_2) = \mathtt{R} \qquad \alpha(\vee,\mathtt{M}_1,\mathtt{M}_2) = \mathtt{M}_1 \vee \mathtt{M}_2$$

Note that for α defined as above we have the following typing rule:

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

where the management of contexts is additive. We would now define $\mathtt{eval}_{\mathcal{M}}$ 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.

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

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

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

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

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

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 $\mathcal{M}$ which decides a language $\mathcal{L}$ the final configuration is either accepting or rejecting hence the term Ext can be applied with the intended meaning.

Lemma 18. *A* decision problem $\mathcal{D} : \{0,1\}^* \to \{0,1\}$ *decidable by an ATM* $\mathcal{M}$ *in polynomial time is programmable in* $\mathrm{STA}_{\mathbf{B}}$.

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

From the well known result of [CKS81] we can conclude.

Theorem 3 (Polynomial Space Completeness)**.** *Every decision problem* $\mathcal{D} \in$ PSPACE *is programmable in* $\mathrm{STA}_{\mathbf{B}}$.

6. Related Topic

In this section we will discuss some topic which are related to the work we have presented.

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

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

and the conditional

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

The obtained system $\mathrm{STA}_\mathbf{W}$ equipped with the obvious reduction relation can be shown to be FPSPACE sound following what we have done for $\mathrm{STA}_\mathbf{B}$. Moreover, analogously to [LM93], completeness for FPSPACE can be proved by considering two distincts data types $\mathbf{S}$ (Church representations of Strings) and $\mathbf{W}$ (Flat words over Booleans) as input and output data type respectively. The above is one of the reasons that leads us to consider $\mathrm{STA}_\mathbf{B}$ instead of the above system.

$\mathrm{STA}_\mathbf{B}$ ***and Soft Linear Logic.*** STA has been introduced as a type assignment counterpart of Soft Linear Logic [Laf04]. $\mathrm{STA}_\mathbf{B}$ is an extension of STA by booleans constants. We now pose our attention to the question of which is the logical counterpart of such extension. We can add to SLL (or define by means of second order quantifier) the additive disjunction $\oplus$ and the rules to deal with it, which in a natural deduction style [RDRR97] are:

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

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

$$\frac{\Gamma \vdash \mathbf{1}}{\Gamma \vdash \mathbf{1} \oplus \mathbf{1}} \ (0) \qquad \frac{\Gamma \vdash \mathbf{1}}{\Gamma \vdash \mathbf{1} \oplus \mathbf{1}} \ (1)$$

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

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

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

We conjecture that the logical system obtained by adding the above modified rule to SLL behaves like $\mathrm{STA}_\mathbf{B}$. In order to prove the polynomial space soundness for such a system we need to mimick in the cut elimination process the abstract machine mechanism of Section 3. For this reason it could be more interesting introduce proof-nets for such a system and study cut elimination in this framework.

A. Technical Proofs

A.1 Proof of Substitution Lemma

The technical notion of height of a variable in a derivation will be useful in the proof of the Substitution Lemma.

Definition 14. *Let* $(\nabla)\ \Gamma, \mathtt{x} : \tau \vdash \mathtt{M} : \sigma$*. The* height *of* $\mathtt{x}$ *in* ∇ *is inductively defined as follows:*

1. *if the last applied rule of* ∇ *is:*

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

then the heigth of $\mathtt{x}$ *in* ∇ *is* 0.

2. *if the last applied rule of* ∇ *is:*

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

then the heigth of $\mathtt{x}$ *in* ∇ *is the max beetween the heights of* $\mathtt{x}_i$ *in* ∇ *for* $1 \leq i \leq k$ *plus one.*

3. *Let* $\mathtt{x} : \tau \in \Gamma$ *and let the last applied rule* π *of* ∇ *be:*

$$\frac{(\Diamond)\ \Gamma \vdash \mathtt{M} : \mathbf{B} \quad (\Box_0)\ \Gamma \vdash \mathtt{N}_0 : A \quad (\Box_0)\ \Gamma \vdash \mathtt{N}_1 : A}{\Gamma, \mathtt{x} : \tau \vdash \ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_0\ \mathtt{else}\ \mathtt{N}_1\ : A}$$

Then the height of $\mathtt{x}$ *in* ∇ *is the max between the heigths of* $\mathtt{x}$ *in* $\Diamond, \Box_0$ *and* $\Box_1$ *respectively plus one.*

4. *In every other case there is an assumption with subject* $\mathtt{x}$ *both in the conclusion of the rule and in one of its premises* $\Diamond$*. Then the height of* $\mathtt{x}$ *in* ∇ *is equal to the height of* $\mathtt{x}$ *in* $\Diamond$ *plus one.*

Proof of Substitution Lemma. By induction on the height of $\mathtt{x}$ in ∇. Base cases (Ax) and (w) are trivial. The cases where ∇ ends by $(\multimap I), (\forall I), (\forall E)$ and $(\multimap E)$ follow directly from the induction hypothesis.
Let ∇ ends by (sp) rule with premise $(\nabla')\ \Gamma', \mathtt{x} : \mu' \vdash \mathtt{M} : \sigma'$ then by Lemma 2.3 $\Diamond \rightsquigarrow \Diamond''$ which is composed by a subderivation ending with an (sp) rule with premise $(\Diamond')\ \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). By the induction hypothesis we have a derivation $(\Box')\ \Gamma', \Delta' \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma'$. By applying the rule (sp) and the sequence of (w) and/or (m) rules we obtain $(\Box)\ \Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$.
Let ∇ ends by

$$\frac{(\nabla_0)\ \Gamma \vdash \mathtt{M}_0 : \mathbf{B} \quad (\nabla_1)\ \Gamma \vdash \mathtt{M}_1 : A \quad (\nabla_1)\ \Gamma \vdash \mathtt{M}_2 : A}{\Gamma \vdash \ \mathtt{if}\ \mathtt{M}_0\ \mathtt{then}\ \mathtt{M}_1\ \mathtt{else}\ \mathtt{M}_2\ : A} \ (\mathbf{B}E)$$

with $\Gamma = \Gamma', \mathtt{x} : \mu$. Then by the induction hypothesis there are derivations $(\Box_0)\ \Gamma', \Delta \vdash \mathtt{M}_0[\mathtt{N}/\mathtt{x}] : \mathbf{B}$, $(\Box_1)\ \Gamma', \Delta \vdash \mathtt{M}_1[\mathtt{N}/\mathtt{x}] : A$ and $(\Box_2)\ \Gamma', \Delta \vdash \mathtt{M}_2[\mathtt{N}/\mathtt{x}] : A$. By applying a $(\mathbf{B}E)$ rule we obtain a derivation $(\Box)$ with conclusion:

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

Let ∇ ends by

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

By Lemma 2.3 $\Diamond \rightsquigarrow \Diamond''$ ending by an (sp) rule with premise $(\Diamond')\ \Delta' \vdash \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). Consider fresh copies of the derivation $\Diamond'$ i.e. $(\Diamond'_j)\ \Delta'_j \vdash \mathtt{N}_j : \mu'$ where $\mathtt{N}_j$ and Δ'_j are fresh copies of $\mathtt{N}$ and Δ' $(1 \leq j \leq m)$.
By induction hypothesis there is a derivation $(\Box_i)\ \Gamma', \mathtt{x}_1 : \mu', \ldots, \mathtt{x}_{i-1} : \mu', \mathtt{x}_{i+1} : \mu', \ldots, \mathtt{x}_m : \mu', \Delta'_i \vdash \mathtt{M}[\mathtt{N}_i/\mathtt{x}_i] : \sigma'$ where the height of $\mathtt{x}_i$ in ∇' is maximal between the heights of $\mathtt{x}_j$ in ∇' for $1 \leq j \leq m$. Since the heights of $\mathtt{x}_j$ in $\Box_i$ continue to be smaller than the height of $\mathtt{x}$ in ∇ we can repeatedly apply induction hypothesis to obtain a derivation $(\Box')\ \Gamma', \Delta'_1, \ldots, \Delta'_m \vdash \mathtt{M}[\mathtt{N}_1/\mathtt{x}_1, \cdots, \mathtt{N}_m/\mathtt{x}_m] : \sigma$. Finally by applying repeatedly the rules (m) and (w) the conclusion follows. $\Box$

A.2 Proof of Weighted Substitution Lemma

It suffices to verify how the weights are modified by the proof of Lemma 3. We will use exactly the same notation as in the lemma.

Proof of Weighted Substitution Lemma. Base cases are trivial and in the cases where ∇ ends by $(\multimap I), (\forall I), (\forall E)$ and $(\multimap E)$ the conclusion follows directly by induction hypothesis.
If ∇ ends by (sp): $\delta(\nabla, r) = r\delta(\nabla', r)$ and $\delta(\Diamond, r) = r\delta(\Diamond', r)$.

By the induction hypothesis $\delta(\Box', r) \leq \delta(\nabla', r) + \delta(\Diamond', r)$ and applying (sp):

$$\delta(\Box, r) \leq r(\delta(\nabla', r) + \delta(\Diamond', r)) = \delta(\nabla, r) + \delta(\Diamond, r)$$

If ∇ ends by $(\mathbf{B}E)$: $\delta(\nabla, r) = \max_{0 \leq i \leq 2}(\delta(\nabla_i, r)) + 1$. By induction hypothesis we have derivations $\delta(\Box_i, r) \leq \delta(\nabla_i, r) + \delta(\Diamond, r)$ for $0 \leq i \leq 2$. and applying a $(\mathbf{B}E)$ rule:

$$\delta(\Box, r) \leq \max_{0 \leq i \leq 2}(\delta(\nabla_i, r) + \delta(\Diamond, r)) = \max_{0 \leq i \leq 2}(\delta(\nabla_i, r)) + \delta(\Diamond, r)$$

If ∇ ends by (m): $\delta(\nabla, r) = \delta(\nabla', r)$ and $\delta(\Diamond, r) = r\delta(\Diamond', r)$. Clearly $\delta(\Diamond', r) = \delta(\Diamond'_j, r)$ so $\delta(\Box', r) \leq \delta(\nabla', r) + m\delta(\Diamond', r)$ and since $r \geq \mathtt{rk}(\nabla)$ then:

$$\delta(\Box', r) \leq \delta(\nabla', r) + r\delta(\Diamond', r) = \delta(\nabla, r) + \delta(\Diamond, r)$$

Now the rules (m) and (w) leave the space weight δ unchanged hence the conclusion follows. $\Box$

References

[AR02] Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 71–79, Washington, D.C., 1989. IEEE Computer Society Press.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier/North-Holland, Amsterdam, London, New York, revised edition, 1984.

[BT04] P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS 2004. IEEE Computer Society*, pages 266–275, 2004.

[CDLRDR05] Paolo Coppola, Ugo Dal Lago, and Simona Ronchi Della Rocca. Elementary affine logic and the call by value lambda calculus. In *TLCA'05*, volume 3461 of *LNCS*, pages 131–145. Springer, 2005.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Gir98] J-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[GKL+07] E. Gädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and its applications*. Springer, 2007.

[Goe92] A. Goerdt. Characterizing complexity classes by higher type primitive recursive definitions. *Theoretical Computer Science*, 100(1):45–66, 1992.

[GR07] M. Gaboardi and S. Ronchi Della Rocca. 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*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.

[Jon01] N.D. Jones. The expressive power of higher-order types or, life without cons. *J. Funct. Program.*, 11(1):55–94, 2001.

[Kri07] J-L. Krivine. A call-by-name lambda calculus machine. *Higher Order and Symbolic Computation*, 2007. To appear.

[Laf04] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.

[LM93] D. Leivant and J-Y. Marion. Lambda calculus characterizations of poly-time. In *Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 1993.

[LM94] D. Leivant and J-Y. Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. In *CSL*, volume 933 of *LNCS*, pages 486–500. Springer, 1994.

[LM97] D. Leivant and J-Y. Marion. Predicative functional recurrence and poly-space. In *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 369–380. Springer-Verlag, 1997.

[Mau03] F. Maurel. Nondeterministic light logics and NP-time. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings*, volume 2701 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2003.

[MR07] V. Mogbil and V. Rahli. Uniform circuits, & boolean proof nets. In Sergei N. Artemov and Anil Nerode, editors, *Proceedings of the Symposium on Logical Foundations of Computer Science (LFCS'07)*, volume 4514 of *Lecture Notes in Computer Science*, pages 401–421. Springer, June 2007.

[Oit01] I. Oitavem. Implicit characterizations of pspace. In *Proof Theory in Computer Science, International Seminar, PTCS 2001, Dagstuhl Castle, Germany, October 7-12, 2001, Proceedings*, volume 2183 of *Lecture Notes in Computer Science*, pages 170–190. Springer, 2001.

[RDRR97] S. Ronchi Della Rocca and L. Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3), 1997.

[Sav70] W. J. Savitch. Relationship between nondeterministic and deterministic tape classes. *JCSS*, 4:177–192, 1970.

[Sch07] U. Schopp. Stratified bounded affine logic for logarithmic space. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 411–420, Washington, DC, USA, 2007. IEEE Computer Society.

[Ter00] K. Terui. Linear logical characterization of polyspace functions (extended abstract), 2000. Unpublished.

[Ter04] K. Terui. Proof nets and boolean circuits. In *LICS '04: Proceedings of the 19nd Annual IEEE Symposium on Logic in Computer Science*, pages 182–191. IEEE Computer Society, 2004.

[Var82] M. Vardi. Complexity and relational query languages. In *Fourteenth Symposium on Theory of Computing*, pages 137–146. ACM, New York, 1982.