Linearity : an analytic tool in the study of complexity and semantics of programming languages

Marco Gaboardi

Dipartimento di Informatica
Università degli Studi di Torino

Ecole doctorale IAEM Lorraine
Institut National Polytechnique de Lorraine

Linearity: an Analytic Tool in the Study of Complexity and Semantics of Programming Languages

THÈSE

présentée et soutenue publiquement le 12 decembre 2007
pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)
et du
Doctorat de l'Universitá degli studi di Torino
(spécialité informatique)
par

Marco Gaboardi

Composition du jury

Rapporteurs:	Jacqueline Vauzeilles	Professeur, Université Paris XIII
	Simone Martini	Professeur, Universitá di Bologna
Examinateurs:	Patrick Baillot	Chargé de Recherche, CNRS
	Paolo Coppola	Professeur, Universitá di Udine
	Jean-Yves Marion	Professeur, ENSMN-INPL
	Simone Martini	Professeur, Universitá di Bologna
	Simona Ronchi Della Rocca	Professeur, Universitá di Torino
	Jacqueline Vauzeilles	Professeur, Université Paris XIII

Dipartimento di Informatica
Università degli Studi di Torino

Ecole doctorale IAEM Lorraine
Institut National Polytechnique de Lorraine

Dottorato di Ricerca in Informatica

Ciclo XX

Linearity: an Analytic Tool in the Study of Complexity and Semantics of Programming Languages

Marco Gaboardi

Supervisori:

Prof.ssa Simona Ronchi Della Rocca

Prof. Jean-Yves Marion

Coordinatore del dottorato:

Prof. Pietro Torasso

Anni Accademici: 2004-2005 2005-2006 2006-2007

Settore scientifico-disciplinare di afferenza: INF/01

Acknowledgments

First and foremost, I express my gratitude to my first supervisor Simona Ronchi Della Rocca. Her invaluable guidance has taught me a lot. Her great enthusiasm and concreteness in doing research have been source of inspiration in my work. Without her help and patience this thesis would not have been possible.
I owe a great debt to my second supervisor Jean-Yves Marion for his support during my staying in Nancy. His suggestions on the directions of my works have been important. Without his guidance a good part of this thesis would never have been written.
I would express my thanks to Jacqueline Vauzeilles and Simone Martini for accepting to be the referees of this thesis and for their valuable comments. I would thank Patrick Baillot and Paolo Coppola for being part of my phd committee. I owe my gratitude to Patrick Baillot also for many helpful comments and suggestions on this thesis. Moreover, he has made it possible my staying in Paris. I enjoyed very much this period spent in interesting and stimulating discussions.
I am in debt with Luca Roversi. His interest in Implicit Computational Complexity and the many stimulating discussions on different topics have greatly influenced me.
I express my thanks to Luca Paolini for having shared with me his ideas that have leaded to the second part of this thesis.
I am grateful to Felice Cardone, my former supervisor, who has brought me to Torino and has always patiently listened and helped me.
I would thank many colleagues for stimulating conversations. In particular, Luca Fossati and the other phd students, Mauro Piccolo and the other members of the Lambda group, Ugo Dal Lago, Paolo Di Giamberardino, Michele Pagani, Romain Péchoux, Colin Riba, Matthieu Kaczmarek, Virgile Mogbil, Damiano Mazza, Gabriele Pulcini.
I would also thank the friends that have made pleasant my staying in Nancy. My particular thanks goes to Noelia, Patrick, Romain, Clémence, Matthieu, Sophie, Julien, Guillem, Heinrich, Colin, Hana, Lucia, Uwe, Clara.
My gratitude goes also to all the people who are a constant presence in my every day life. In particular, I am extremely grateful to my family. They are a constant source of backing and help. Finally, my special thanks goes to Anne. She has shared with me these three years, giving me her invaluable understanding and support.

Résume : Dans la première partie, on propose un système de type pour le lambda-calcul, dans le style du calcul des séquents, nomme " Soft Type Assignment " (STA) qui est inspiré par la logique linéaire " soft ". STA a la propriété de réduction du sujet et est correct et complète pour les calculs en temps polynomial. Par la suite on propose un déduction naturelle, $\mathrm{STA}_{\mathrm{N}}$. Ce système est simple mais il a le désavantage que les variables dans le sujet peuvent être explicitement renommées. Pour résoudre ce problème, on propose le système $\mathrm{STA}_{\mathrm{M}}$, où les contextes sont des multi-ensembles, donc les règles pour renommer les variables peuvent être interdit. L'inférence de type pour $\mathrm{STA}_{\mathrm{M}}$ ne semble pas décidable. On propose un algorithme $\mathbf{\Pi}$ qui pour chaque lambda-terme rend l'ensemble de contraintes que doivent être satisfait pour que le terme soit type. $\mathbf{\Pi}$ est correct et complet. Ensuite on étend le lambda-calcul par des constantes booléennes et on propose le système $\mathrm{STA}_{\mathbf{B}}$. La particularité de $\mathrm{STA}_{\mathbf{B}}$ est que la règle du conditionnel utilise les contextes de façon additive. Chaque programme de $\mathrm{STA}_{\mathbf{B}}$ peut être exécuté, par une machine abstraite, en espace polynomial. De plus le système est aussi complet pour PSPACE. Dans la deuxième partie, on propose une restriction de PCF, nommée $\mathscr{SL}$PCF. Ce langage est équipé avec une sémantique opérationnelle qui mélange l'appelle par nom et l'appelle par valeur et peut être interprèté en mode standard dans les espaces cohérents linéaires. $\mathscr{SL}$PCF est complet pour les fonctions récursives, mais il n'est pas complet et donc il n'est pas fully abstract pour les espaces cohérents linéaires.
Mot clés : complexité implicite, lambda-calcul, logique linéaire, sémantique denotationelle.

Abstract: In the first part, we propose, inspired by Soft Linear Logic, a type assignment system for lambda-calculus in sequent calculus style, named Soft Type Assignment (STA). STA enjoys the subject reduction property. and is correct and complete for polynomial time computations. Then, we propose a natural deduction named $\mathrm{STA}_{\mathrm{N}}$. While simple, $\mathrm{STA}_{\mathrm{N}}$ has the disadvantage of allowing the explicit renaming of variables in the subject. To overcome to this problem, we propose another natural deduction system, named $\mathrm{STA}_{\mathrm{M}}$, where contexts are multisets, hence rules renaming variables can be avoided. The type inference for $\mathrm{STA}_{\mathrm{M}}$ seems in general undecidable. We propose an algorithm $\mathbf{\Pi}$ returning, for every lambda-term, a set of constraints that need to be satisfied in order to type the term. $\mathbf{\Pi}$ is correct and complete. We extend the lambda-calculus by basic boolean constants and we propose the system $\mathrm{STA}_{\mathbf{B}}$. The peculiarity of $\mathrm{STA}_{\mathbf{B}}$ is that the conditional rule treats the contexts in an additive way. Every $\mathrm{STA}_{\mathbf{B}}$ program can be executed, through an abstract machine, in

polynomial space. Moreover, $\mathrm{STA}_{\mathbf{B}}$ is also complete for PSPACE. In the second part we propose a restriction of PCF, named $\mathcal{SL}$PCF. The language is naturally equipped with an operational semantics mixing call-by-name and call-by-value parameter passing and it can be interpreted in linear coherence space in a standard way. $\mathcal{SL}$PCF is recursive complete, but it is not complete, and thus not fully abstract, with respect to linear coherence spaces.
Key words : implicit computational complexity, lambda-calculus, linear logic, denotational semantics.

Contents

Résumé substantiel

Dans cette thèse, on analyse comment le concept de *linéarité* peut être utilisé pour étudier des aspects des langages de programmation. Pour *linéarité* nous voulons considérer différents concepts en relation à les idées sur la logique linéaire de Girard [Girard, 1987].
La grand innovation de la logique linéaire par rapport à la logique classique et intuitionniste est qui permet contrôler les ressources. Pour cette raison, la logique linéaire a attiré différents chercheurs et a inspiré plusieurs travaux de recherche dans le domaine des langages de programmation.
In cette thèse, on considère, en particulier, deux aspects très importants de langages de programmation : la *complexité implicite* et la *sémantique denotationelle*.

Dans la première partie, on étudie différents problèmes liés à la complexité implicite des langages de programmation.
En premier, on étudie le problème de concevoir un langage de programmation correct et complète pour les computations en temps polynomial. On considère l'emploi du λ-calcul comment paradigme abstraite des langages de programmation et l'emploi des types pour caractériser les propriétés de complexité.
On propose un système de type, dans le style du calcul des séquents, nomme " Soft Type Assignment " (STA) qui est inspiré par la logique linéaire " soft ", une logique qu'il caractérise PTIME. L'ensemble des types de STA est un sous-ensemble des formules de la logique linéaire soft, où la modalité ! ne peux pas apparaître dans la partie droite d'un flash. Donc, l'application de certaines règles est restreinte aux types linéaires. *STA* a la propriété de réduction du sujet. Cette propriété est important parce que le typage est nécessaire pour assurer le borne sur le réductions des termes. STA est correct et complète pour les calculs en temps polynomial. En particulier, les termes que peuvent être type in STA peuvent être réduit à la forme normale avec un nombre polynomial d'étapes de réduction. En plus, ces termes sont suffisants pour coder les machines de Turing polynomiales.

Les systèmes de type dans le style du calcul des séquents sont normalement difficiles à utiliser. Pour cette raison, on étudie le problème de dessiner une version en déduction naturelle de STA.
En premier, on propose un simple système en déduction naturelle, nomme $\mathrm{STA}_{\mathrm{N}}$. On prouve que $\mathrm{STA}_{\mathrm{N}}$ est équivalent a STA et il a le même propriété de complexité. $\mathrm{STA}_{\mathrm{N}}$ est simple mais il a le désavantage que les variables dans le sujet peuvent être explicitement renommées.
Pour résoudre ce problème, on propose un autre système en déduction naturelle, nomme $\mathrm{STA}_{\mathrm{M}}$, où les contextes sont des multi-ensembles. Les multi-ensembles permettent de réutiliser les nomes, donc les règles pour renommer les variables peuvent être interdit. On démontre que les prouves de $\mathrm{STA}_{\mathrm{M}}$ sont en correspondance direct avec les prouves de $\mathrm{STA}_{\mathrm{N}}$ et que les deux systèmes ont les même propriété de complexité.

Le pas suivant aux travaux précédents, c'est étudier le problème d'inférence des types pour le système $\mathrm{STA}_{\mathrm{M}}$. A cause du quantificateur du deuxième ordre, l'inférence de type pour $\mathrm{STA}_{\mathrm{M}}$ ne semble pas décidable.
On propose un algorithme $\mathbf{\Pi}$ qui pour chaque lambda-terme rend un ensemble de contraintes que doivent être satisfait pour typer le terme dans $\mathrm{STA}_{\mathrm{M}}$. On démontre que $\mathbf{\Pi}$ est correct : étant donné un terme, si les contraintes produites par $\mathbf{\Pi}$ sont satisfaisables, un typage pour le terme existe. En plus on démontre que $\mathbf{\Pi}$ est aussi complet : pour chaque terme, si le terme peut être typé, alors l'ensemble des contraintes produites par $\mathbf{\Pi}$ est satisfaisable. La satisfiabilité de ce genre de contraintes en général n'est pas décidable, donc une restriction du système est nécessaire.

Pour terminer la première partie de la thèse on étudie le problème de concevoir un langage de programmation correct et complet pour les computations polynomiales. Une façon naturelle pour étudier ce problème est considérer comment étendre STA pour caractériser PSPACE.
On considère un langage qui étend le lambda-calcul par des constantes booléennes et on propose un système $\mathrm{STA}_{\mathbf{B}}$, qui étend la version en déduction naturelle $\mathrm{STA}_{\mathrm{N}}$ de STA par des règles pour les booléens.

La particularité de $\mathrm{STA}_{\mathbf{B}}$ est le traitement des contextes dans la règle du conditionnel. En particulier, la règle du conditionnel utilise les contextes de façon additive, quand les autre règles, comment dans $\mathrm{STA}_{\mathbf{N}}$, utilise les contextes de façon multiplicatives.
$\mathrm{STA}_{\mathbf{B}}$ est équipé avec une sémantique opérationnelle donné par une machine abstraite à grands étapes. Ce machine est utile pour démontrer $\mathrm{STA}_{\mathbf{B}}$ correct pour les computation polynomiales. Effectivement, chaque programme de $\mathrm{STA}_{\mathbf{B}}$ peut être exécuté, par la machine abstraite, en espace polynomial dans la taille de l'entrée.
Les machines de Turing alternantes travaillant en temps polynomial peuvent être facilement représentées par des termes qui peuvent être typés dans $\mathrm{STA}_{\mathbf{B}}$. Donc, par un résultat bien connu à propos des machines de Turing alternantes, le problème de décision en espace polynomial peut être programmé en $\mathrm{STA}_{\mathbf{B}}$, et le système est complet pour PSPACE.

Dans la deuxième partie, on étudie comment concevoir un langage " fully abstract " pour une interprétation denotationelle dans une model linéaire spécifique. Le model que on considère est le fragment des espaces cohérents de Girard, comprenant le domaine infini des entiers naturels et le constructeur d'espace des fonctions linéaires, comment unique constructeur d'espace cohérents.
On propose un nouveau langage, nommée $\mathcal{SL}\mathrm{PCF}$, que est une restriction de PCF. $\mathcal{SL}\mathrm{PCF}$ est équipé avec une sémantique opérationnelle qui mélange l'appelle par nom et l'appelle par valeur et peut être interprété en mode standard dans les espaces cohérents linéaires. On prouve que l'interprétation est correcte par rapport à la sémantique opérationnelle, donc dans $\mathcal{SL}\mathrm{PCF}$, on peut programmer seulement des fonctions du modèle linéaire considéré.
$\mathcal{SL}\mathrm{PCF}$ est complet pour les fonctions récursives, mais il n'est pas complet et donc il n'est pas fully abstract pour les espaces cohérents linéaires. Donc une extension est nécessaire.

Extended Abstract

In this thesis we analyze how the concept of *linearity* can be fruitfully exploited in studying the properties of programming languages. By the term *linearity*, we in fact refer to different concepts, more or less related to the ideas underlying Girard's Linear Logic [Girard, 1987].
The great innovation of linear logic with respect to classical or intuitionistic logics is that the former gives the opportunity to deal with resources. For this reason, since its origins, linear logic has attracted many researchers and has inspired different works in programming language design.
In this thesis we consider, in particular, two important aspects of programming languages: *implicit computational complexity* and *denotational semantics*.

In the first part of the thesis we study some problems related to the implicit computational complexity of programming languages.
The first step is an examination of the problem of designing a programming language correct and complete for polynomial time computations. We consider the use of λ-calculus as an abstract paradigm of programming languages and the use of types to characterize program complexity properties.
We propose a type assignment system in sequent calculus style, named Soft Type Assignment (STA). This system is inspired by Soft Linear Logic, a logical system characterizing PTIME. The set of STA types is a subset of Soft Linear Logic formulae, where the modality ! is not allowed in the right hand side of an arrow. Consequently, the applications of some rules are restricted to linear types.
STA enjoys the subject reduction property. This specific property is of great importance, since typing is intended to assure the complexity bound on terms reductions. STA is correct and complete for polynomial time computations. Indeed, the typable terms have a polynomial number of β-reductions to normal form. Moreover, they are sufficient to encode polynomial time Turing machines.

Type assignment systems in sequent calculus style are usually difficult to manage. For this reason, we study the problem of designing a natural deduction version of the Soft Type Assignment system STA.
The first solution we propose is a simple natural deduction system named $\mathrm{STA}_{\mathrm{N}}$. We prove that, besides being equivalent to STA, it also enjoys the same complexity properties. While simple, $\mathrm{STA}_{\mathrm{N}}$ has the disadvantage of allowing the explicit renaming of variables in the subject.
To overcome this problem, we propose another natural deduction system, named $\mathrm{STA}_{\mathrm{M}}$. We prove that $\mathrm{STA}_{\mathrm{M}}$ derivations are in one to one correspondence with $\mathrm{STA}_{\mathrm{N}}$ derivations. Hence, $\mathrm{STA}_{\mathrm{M}}$ enjoys the same complexity properties as $\mathrm{STA}_{\mathrm{N}}$. The idea underlying $\mathrm{STA}_{\mathrm{M}}$ is to use multisets as contexts. A multiset allows to reuse variable names, hence rules for renaming variables can be avoided.

As a natural continuation of the previous work we look at the type inference problem for the system $\mathrm{STA}_{\mathrm{M}}$. Due to the presence of second order quantifier, it seems in general undecidable.
We propose an algorithm $\mathbf{\Pi}$ such that, for every λ-term, returns a set of constraints. These constraints need to be satisfied in order to type the term in $\mathrm{STA}_{\mathrm{M}}$. We prove that $\mathbf{\Pi}$ is correct in the sense that, given a term, if the generated constraints are satisfiable, a typing for it exists. Moreover, we prove that $\mathbf{\Pi}$ is also complete in the sense that, for every typable term, the set of constraints generated by $\mathbf{\Pi}$ is satisfiable.
Satisfiability of constraints of this kind is in general undecidable. Consequently, some restriction of the system seems necessary.

To conclude the first part of the thesis we look at the problem of designing a programming language correct and complete for polynomial space computations. The most natural way to study this problem is by considering how STA can be extended in order to characterize PSPACE.
We consider a language extending the λ-calculus by basic boolean constants and we propose a type assignment system, $\mathrm{STA}_{\mathbf{B}}$, extending the natural deduction version $\mathrm{STA}_{\mathrm{N}}$ of STA by rules for booleans.

The peculiarity of $\mathrm{STA}_{\mathbf{B}}$ is the treatment of contexts in the conditional rule. The conditional rule treats the contexts in an additive way. In every other rule, on the other hand, contexts are treated in a multiplicative way, as in $\mathrm{STA}_{\mathbf{N}}$.
$\mathrm{STA}_{\mathbf{B}}$ is equipped with an operational semantics based on a big step abstract machine. This is a useful tool in order to prove correctness for polynomial space computations. Every $\mathrm{STA}_{\mathbf{B}}$ program can in fact be executed, by the abstract machine, in polynomial space in the size of the input.
Alternating Turing machines working in polynomial time are easily representable by terms typable in $\mathrm{STA}_{\mathbf{B}}$. Hence, by a well known result on alternating Turing machines, polynomial space decision problems can be programmed in $\mathrm{STA}_{\mathbf{B}}$. The system is thus complete for PSPACE.

In the second part of the thesis we study the problem of designing a programming language fully abstract with respect to a denotational interpretation in a specific linear model. The model we consider is the fragment of Girard's coherence spaces, including the infinite flat domain representing natural numbers, and the linear function space constructor as the only coherence space constructor.
We propose a new language, named $\mathcal{SL}\mathrm{PCF}$. Such language is a restriction of PCF. $\mathcal{SL}\mathrm{PCF}$ is naturally equipped with an operational semantics mixing call-by-name and call-by-value parameter passing and it can be interpreted in linear coherence space in a standard way. We prove that the interpretation is correct with respect to the operational semantics. This means that $\mathcal{SL}\mathrm{PCF}$ is able to programme only functions of the considered linear model.
$\mathcal{SL}\mathrm{PCF}$ is recursive complete. Nevertheless, it is not complete, and thus not fully abstract, with respect to linear coherence spaces. An extension is therefore necessary.

Part I.

Linearity and Complexity

In this part of the thesis we consider some problems that fit in the *Implicit Computational Complexity* research line.
Computational complexity, since its birth in the 70's, has attracted the attention of many computer scientists both from the theoretical and practical points of view. From a theoretical perspective, computational complexity is a step towards the better understanding of the notion of computability, its properties and its limits. From a practical perspective, its interest lies in the possibility to improve, predict and guarantee the resource usage of software. This is an important step in the development of more efficient and correct programs.
The Implicit Computational Complexity research line aims at the study of computational complexity classes through machine independent characterizations, in particular recursion-theoretic and logical characterizations. From a theoretical perspective, implicit computational complexity tries to understand the logical and recursion-theoretic principles usually hidden in the definitions of complexity classes. From a practical perspective, it aims at the design of systems and tools that can be useful for the static verification and certification of resource usage of programs.

The starting point of Implicit Computational Complexity is the seminal paper of Cobham [Cobham, 1964]. In this work the author gives a characterization of the class of functions computable by Turing machines in polynomial time in the size of the input (FPTIME) through a class of recursive functions based on a particular recursion scheme called *bounded recursion on notation*. Cobham's characterization of functions computable in polynomial time, while being the first machine independent characterization of a complexity class, has the disadvantage of involving a resource bound (in the form of a bound on the result size) in the definition of the system.
Further important developments in the same direction date back to the eighties and nineties, in a series of works characterizing different complexity classes through bounded recursion schemas. See [Clote, 1995] for a survey.
In [Bellantoni and Cook, 1992], the authors give a characterization of FPTIME through a recursion schema, called *predicative recursion on notation*, which takes into account an explicit stratification over data. The use of stratification over data permits to avoid explicit bounds. A similar result has been independently founded in [Leivant, 1991].
Following the ideas of Bellantoni, Cook and Leivant many other complexity classes have been characterized through recursion schema taking into account data stratification. See for instance [Bellantoni, 1992], [Leivant and Marion, 1994], [Oitavem, 2001] and [Bonfante et al., 2006]. The same principle of data stratification has been used for

characterizations of complexity classes through typed lambda calculus systems in [Leivant and Marion, 1993] and [Leivant and Marion, 1997].
In [Hofmann, 2000] and [Hofmann, 2003] the author proposes another interesting approach to the characterization of complexity classes. Polynomial time computations are characterized by using a type system including a resource type and recursion operators. In this approach, the data stratification is no more necessary.

In parallel with the development of the recursion-theoretic approach, another approach to the characterization of complexity classes that has attracted more and more attention is the design of logical systems with intrinsic computational complexity properties based on linear logic.
One of the innovations of linear logic with respect to classical or intuitionistic logic is the explicit control of structural rules, i.e. contraction, weakening, exchange. A careful treatment of the contraction rule permits to control the complexity of the proof normalization process. This feature allows the use of Linear Logic subsystems in order to obtain purely logical characterizations of complexity classes.
This is achieved by designing logical systems around the proofs-as-programs paradigm, where proofs correspond to algorithms and the normalization process corresponds to algorithm execution. Therefore, limiting the complexity of proof normalization corresponds to limiting the computational complexity. This approach is in contrast with previous logical approaches to the characterization of complexity classes where the characterizations are in terms of provability of certain restricted formulas. See [Immerman, 1999] for a wide treatment of this subject.
In [Girard et al., 1992] the authors propose Bounded Linear Logic (BLL), a first attempt in designing a subsystem of linear logic characterizing a particular complexity class. Bounded Linear Logic is a subsystem of second order linear logic corresponding to the complexity class FPTIME. Each BLL proof is normalizable, via cut-elimination, in polynomial time in the size of the proof, and each function computable in polynomial time by a Turing machine is representable by a proof in BLL. The drawback of this proposal is that size bounds appear explicitly in BLL modalities. Nevertheless, it should be noted that the preservation of the complexity bound is intrinsic in the logic itself.
A more recent proposal of a system capturing FPTIME is Light Linear Logic (LLL) [Girard, 1998]. In LLL the complexity bound on the proofs normalization, via cut-elimination, is assured only by restricting contraction. Hence, no explicit resource bound appears in the modalities. Nevertheless, a new modality is needed in order to

gain the necessary expressivity.
BLL and LLL have shown the effectiveness of the linear logic approach to the characterization of complexity class. As a result, during the last years, many different subsystems of linear logic have been proposed in order to characterize different complexity classes. Some of them are: Soft Linear Logic (SLL) [Lafont, 2004] and Elementary Linear Logic (ELL) [Girard, 1998], Intuitionistic Light Affine Logic (ILAL) [Asperti and Roversi, 2002], non deterministic Intuitionistic Light Affine Logic (nILAL) [Maurel, 2003] and Stratified Bounded Affine Logic (SBAL) [Schopp, 2007]. In what follows all these systems will be referred to as light logics.
Since the discovery of light logics, different authors have worked in order to study their properties, possible extensions and possible restrictions. The collection of these works brought to a well-established theory. Nevertheless, many important problems are waiting for an answer.

In this part of the thesis we analyze in more depth the problem of designing programming languages which characterize complexity classes using properties of light logics, in particular of Soft Linear Logic. In what follows, we outline and motivate the content of this part of the thesis.

Chapter 1: Soft Linear Logic

Soft Linear Logic is a logical counterpart of polynomial time computations. Its proofs are normalizable, via cut-elimination, in polynomial time in the size of the proofs. Moreover, each decision problem computable in polynomial time by a Turing machine is representable by a proof in Soft Linear Logic. Hence, Soft Linear Logic characterizes the complexity class of decision problems computable in polynomial time by deterministic Turing machines (PTIME).
In this chapter we recall the syntax and the rules, in sequent calculus, of Soft Linear Logic and consider some of the logical laws underlying it Finally, we state formally the complexity properties which permit the characterization of the complexity class.

Chapter 2: A Soft Type Assignment System for λ-calculus

In this chapter we study how to use Soft Linear Logic for the design of a programming language with intrinsic polynomial time complexity bound, through the proofs-as-programs correspondence.
A way to do this is by considering a term language simulating the logical principles of the system in the syntax, and by decorating the logical rules by the constructs

of such a specific term language. This approach has been followed for example in [Baillot and Mogbil, 2004] for SLL and in [Benton et al., 1993] and [Terui, 2001] for LL and ILAL respectively. Unfortunately, due to the presence of modalities, the resulting language of [Baillot and Mogbil, 2004] has a complex syntactical structure that makes the programming task awkward.
Here we follow a different approach. We fix as starting points:

1. The use of λ-calculus as an abstract paradigm of programming languages.

2. The use of types to characterize program properties.

The objective is the design of a type assignment system for λ-calculus, where types are formulae of Soft Linear Logic, in such a way that the logical properties are inherited by the well typed terms. In this way types can be used for statically checking, beside the usual notion of correctness, also the property of having polynomial time complexity.
The effectiveness of this approach has been shown by Baillot and Terui in [Baillot and Terui, 2004], where they propose Dual Light Affine Logic (DLAL), a type assignment for λ-calculus correct and complete with respect to polynomial time computations. DLAL corresponds to a fragment of Light Affine Logic [Asperti, 1998], using as types a language with a linear and intuitionistic type arrow, and one modality. We consider the design of a type assignment system for λ-calculus correct and complete with respect to polynomial time computations inspired by Soft Linear Logic. The motivation is twofold.
The design of λ-calculi with implicit complexity properties is a very new research line. Therefore, exploring different approaches is necessary in order to compare them. In particular, SLL formulae look simpler than LAL ones since they have just one modality. LAL has two modalities, and this could in principle give rise to a greater easiness of the system. Moreover, SLL has been proved to be complete just for PTIME, while we want a type assignment system complete for FPTIME: we thus prefer to explore at the same time if it is possible, when switching from formulae to types, to prove this further property.

We start from the original version of second order SLL with the only connective $\multimap$, the modality ! and the quantifier $\forall$, given in sequent calculus style. The main problem in designing the desired type assignment system is that, in general, in a modal logic setting, the good properties of proofs are not easily inherited by λ-terms.
In particular, there is a mismatch between β-reduction in the λ-calculus and cut-elimination in logical systems, which makes it difficult to obtain the subject reduction

property. Here, subject reduction is an important property since we would use typing to assure the complexity bound on terms reductions. Moreover, this mismatch makes it difficult to inherit the complexity properties from the logic, as discussed in [Baillot and Terui, 2004]. To solve these problems we make use of the fact that, in a "naive" decorated sequent calculus, there is a redundancy of proofs, in the sense that the same λ-term can arise from different proofs.
Therefore, we propose a restricted system called Soft Type Assignment (STA). In this system the set of derivations corresponds to a proper subset of proofs in the affine version of Soft Linear Logic. Types are a subset of SLL formulae where, in the same spirit as [Baillot and Terui, 2004], the modality ! is not allowed in the right hand of an arrow and the applications of some rules, in particular the *cut* rule, are restricted to linear types.
STA enjoys subject reduction, and the good properties of SLL can be adapted to the case of STA. We prove that λ-terms typable in STA reduce to normal form in polynomial time. Moreover, we prove that the language of typable terms is complete both for PTIME and FPTIME. In fact, polynomial time functions can be computed, through a simulation of polynomial time deterministic Turing machines, by terms typable in STA.
The content of this chapter is an extended version of the works appeared in [Gaboardi and Ronchi Della Rocca, 2006], [Gaboardi and Ronchi Della Rocca, 2007].

Chapter 3: Soft Type Assignment system in Natural Deduction style

In the previous chapter we have introduced the Soft Type Assignment system STA in sequent calculus style. Here we consider the problem of designing a natural deduction type assignment system corresponding to STA.
Light Logics are usually presented in sequent calculus style, because sequent calculus is considered an appropriate framework to study structural properties. Moreover, since cut elimination steps correspond to atomic computation steps, sequent calculus seems particularly interesting for studying complexity properties.
On the other hand, type assignment systems for λ-calculus are usually presented in natural deduction style because derivation trees in natural deductions correspond more directly to the abstract syntax trees of terms, making natural deduction derivations easier to manage. Furthermore, proof normalization steps correspond directly to β-reduction. See [Barendregt and Ghilezan, 2000] for an interesting discussion about this topic.

We here propose the $\mathrm{STA}_{\mathrm{N}}$ system, which is a simple Natural Deduction system corresponding in an intuitive way to STA. The proof that the two systems are equivalent can be obtained in a standard way. Nevertheless, since we are interested in the preservation of complexity measures, we prefer to take two subsequent steps. Firstly we prove the equivalence at the typability level. Then, we show that the measures defined over STA derivations can be easily adapted to $\mathrm{STA}_{\mathrm{N}}$ derivations. In this way, we can prove that each $\mathrm{STA}_{\mathrm{N}}$ derivation enjoys the same measures of the corresponding STA derivation, and vice-versa.

$\mathrm{STA}_{\mathrm{N}}$ is simple and intuitive, but it has the drawback that the multiplexor rule allows the renaming of variables in the subject. This fact, while being essential for $\mathrm{STA}_{\mathrm{N}}$ expressivity, prevents the possibility of reasoning by induction on the structure of terms. Moreover, this makes the task of designing a type inference algorithm more difficult. For this reason, we introduce another natural deduction system, named $\mathrm{STA}_{\mathrm{M}}$.

Contexts in $\mathrm{STA}_{\mathrm{M}}$ are multisets of type assignments. We make use of multisets to trace the different uses of variable names. This makes the renaming unnecessary. Hence, as usual for natural deduction systems, subjects in $\mathrm{STA}_{\mathrm{M}}$ are built just by the axiom and arrow introduction, and by elimination rules.

The use of multisets could seem unpleasant, nevertheless a multiset can be collapsed into a set by using the multiplexor rule repeatedly. In fact, we prove that $\mathrm{STA}_{\mathrm{M}}$ derivations are in one-to-one correspondence with $\mathrm{STA}_{\mathrm{N}}$ derivations. Hence, the two systems enjoy the same complexity properties.

The use of multisets to trace different uses of variable names has been already used in some intersection types systems. In particular, a treatment of the contexts similar to the one presented here has also been used in [Coppo et al., 1980] and [de Carvalho, 2007]. It could be interesting to study the relations between $\mathrm{STA}_{\mathrm{M}}$ and these works.

Chapter 4: A Taste of Type Inference for STA

In this chapter we examine the problem of type inference for the Soft Type Assignment system. In Chapter 2, in designing the Soft Type Assignment system, we have used λ-calculus as language and types to characterize complexity properties of terms. An important aspect of this approach is that types allow to statically check the program properties of interest. This approach is particularly suitable when there is an effective and efficient automatic procedure for checking if a term is typable. The problem of deciding if a term is typable in a given system is usually referred to as the type inference problem.

The decidability of the type inference problem for the simply typed λ-calculus has been

shown independently in [Curry, 1969], [Hindley, 1969] and [Milner, 1978]. Moreover, it has been shown that for this system the type inference can be performed efficiently. These results are the core of the well-established theory of type inference for the ML language.

In the presence of the second order quantifier the situation gets more complicated. The type inference problem for System F has been shown undecidable in [Wells, 1994] and [Wells, 1999]. Nevertheless, since System F is interesting both from the theoretical and practical point of view, different partial solutions have been proposed, e.g. [Giannini and Ronchi Della Rocca, 1994], [Kfoury and Wells, 1994] and [Rémy, 2005]. Here we are particularly interested in [Giannini and Ronchi Della Rocca, 1994], where the authors propose a stratification of System F in a countable set of type assignment systems with decidable type inference.

The possibility of statically checking complexity properties of programs is particularly appealing. For this reason, different authors have considered the type inference problem for light logics. Unfortunately, type assignment systems inspired by light logics usually include the second order quantifier. The undecidability of type inference for System F has brought different authors to consider only the propositional fragments of these logics.

The type inference problem for the propositional fragment of LAL has been studied in [Baillot, 2002] and [Baillot, 2004]. The author, in [Baillot, 2004], has shown that such problem is decidable by reducing it to a problem of satisfiability for a system of inequalities on words over a binary alphabet.

In [Coppola and Martini, 2001], the type inference problem for the propositional fragment of EAL has been shown decidable. Moreover, different algorithms performing type inference in EAL have been proposed in [Coppola and Ronchi della Rocca, 2003], [Coppola et al., 2005] and [Baillot and Terui, 2005]. The interest in type inference for EAL is also due to the fact that typable terms can be evaluated by optimal reduction much more easily than general terms, see [Coppola and Martini, 2006] and [Baillot et al., 2007].

An approach not limited to the propositional fragment of a light logic has been taken in [Atassi et al., 2006]. The authors have therein considered the problem of deciding if a System F term can be decorated in a DLAL typed term. They have shown that there is an effective procedure to do this and, moreover, that the decoration can be performed in polynomial time.

Subsequently, we here tackle the type inference problem for the Soft Type Assignment

system. We consider the entire system including second order quantifier and we follow what has been done in [Giannini and Ronchi Della Rocca, 1994] for System F.
We consider the multiset natural deduction system $\mathrm{STA}_{\mathrm{M}}$ which was proved in Chapter 3 to be equivalent to STA. Furthermore, we propose an algorithm $\mathbf{\Pi}$ for $\mathrm{STA}_{\mathrm{M}}$. For every λ-term, $\mathbf{\Pi}$ returns the set of constraints that need to be satisfied in order to type the term.
We prove that $\mathbf{\Pi}$ is correct: given a term $\mathtt{M}$, if the constraints generated by $\mathbf{\Pi}(\mathtt{M})$ are satisfiable then a typing for $\mathtt{M}$ exists. Moreover, we prove that $\mathbf{\Pi}$ is also complete: for every typable term $\mathtt{M}$, the set of constraints returned by $\mathbf{\Pi}(\mathtt{M})$ is satisfiable.
The type inference problem for $\mathrm{STA}_{\mathrm{M}}$ can indeed be reformulated as the problem of deciding if a substitution satisfying the constraints generated by $\mathbf{\Pi}$ exists. In [Giannini and Ronchi Della Rocca, 1994], the authors show that the problem of satisfiability of this kind of constraints is equivalent to the semi-unification problem, and hence undecidable.
Thus, following what has been done in [Giannini and Ronchi Della Rocca, 1994] for System F, in future works we will stratify $\mathrm{STA}_{\mathrm{M}}$ in a countable set of type assignment systems $\mathrm{STA}^n_{\mathrm{M}}$ with decidable type inference. In this way, we hope to obtain a system for polynomial time computation with decidable type inference.

Chapter 5: A logical account of PSPACE

In this chapter we study the problem of designing a programming language, inspired by light logics, correct and complete for the class of decision problems computable by deterministic Turing machines in polynomial space (PSPACE).
The main motivation is to show that the light logic approach can be fruitfully used in the characterization of different complexity classes. While there are different light logic systems characterizing complexity classes, none of them characterizes polynomial space complexity classes.
In the literature there are some implicit characterizations of polynomial space computations. The characterizations in [Leivant and Marion, 1994], [Leivant and Marion, 1997] and [Oitavem, 2001] 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 [Grädel et al., 2007]. Finally, there are some algebraic characterizations like the ones in [Goerdt, 1992] or in [Jones, 2001]. Neverthelees these are, in essence, over finite domains.
None of the above characterizations is based on the light logics approach. Indeed, the

only proposal for a characterization of PSPACE with a light system has been made by Terui in [Terui, 2000], but the work has never been completed.
Characterizing space complexity classes through the light logics approach seems a difficult task. In fact, the control of the cut elimination procedure is a direct way to control execution time, but its relations with execution space are not so clear.
Only recently, the first characterization of a space complexity class through a light logic has appeared. In [Schopp, 2007], the author proposes Stratified Bounded Affine Logic (SBAL), a logical system characterizing logarithmic space computations. In the characterization, the difficult part is to achieve logarithmic space soundness. In order to do this, the author considers only proofs of certain sequents to represent the functions in logarithmic space. The author interprets such proofs in a space-efficient game semantic model.

In this chapter, the same approach followed in Chapter 2 is used for studying polynomial space complexity classes. In particular, we consider as starting points the use of a λ-calculus like language and the use of types to guarantee, besides the functional correctness, also complexity properties. This is done in order to use types in a static way to check the correct behaviour of the programs, also with respect to the resource usage.
A well-known fact, in computational complexity, is that polynomial space deterministic Turing machine computations and polynomial space non-deterministic Turing machine computations (NPSPACE) both coincide with polynomial time alternating Turing machine computations (APTIME) [Savitch, 1970, Chandra et al., 1981]. In particular,

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

Inspired by the above result, we consider possible extensions to the type assignment STA for λ-calculus presented in Chapter 2.
We propose $\text{STA}_{\mathbf{B}}$ a type assignment system, 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 booleans is the following:

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

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

A similar rule has been proposed by Maurel in his non deterministic Intuitionistic Light Affine Logic (nILAL) [Maurel, 2003] in order to characterize non deterministic polynomial time. More precisely nILAL introduces an explicit sum rule to deal with non deterministic computations.

While the polytime soundness result for STA is not related to a particular evaluation strategy, here, for characterizing space complexity, the evaluation should be done carefully. Indeed, a wrong evaluation can bring to exponentially sized terms. We thus define a call-by-name evaluation machine, inspired by Krivine's machine [Krivine, 2007] for λ-calculus, where substitutions are made only on head variables. This machine is equipped with a memory device which allows to easily determine the space used, as the dimension of the maximal machine configuration.

We prove that, if the machine takes as input a program, i.e. a closed term typable with a ground type, then the size of each configuration is bounded in a polynomial way in the size of the input. Every program is therefore evaluated by the machine in polynomial space. On the other hand, we encode every polynomial time alternating Turing machine by a program of $\mathrm{STA}_{\mathbf{B}}$. The simulation relies on a higher order representation of a parameter substitution recurrence schema analogous to the one introduced in [Leivant and Marion, 1994].

The content of this chapter will appear in a concise version in [Gaboardi et al., 2008].

1. Soft Linear Logic

In this chapter we recall Soft Linear Logic [Lafont, 2004]. Soft Linear Logic has been introduced by Lafont as: "a subsystem of second-order linear logic with restricted rules for exponentials so that proofs correspond to polynomial time algorithms, and vice-versa." We present SLL in a sequent calculus version. We consider some of the logical laws underlying it. Finally we recall the complexity properties which permit the characterization of PTIME.

1.1. Soft Linear Logic syntax

We introduce the intuitionistic second order version of Soft Linear Logic in a sequent calculus setting.

Definition 1. *The formulae of* Second order Intuitionistic Soft Linear Logic (SLL) *are inductively defined as:*

- *propositional variables* $\alpha, \beta, \ldots$ *and the constant* $\mathbf{1}$ *are formulae*
- *if* A *and* B *are formulae, then so are* $A \multimap B$, $A \otimes B$, $A \& B$ *and* $!A$
- *if* A *is a formula and* α *is a propositional variable, then* $\forall \alpha.A$ *is a formula*

Connectives and modalities of SLL can be classified in the following way. $\multimap$ and $\otimes$ are *multiplicative* connectives, $\&$ is an *additive* connective and ! is an *exponential*. Moreover $\mathbf{1}$ is the multiplicative unit. Formulae of SLL are classified according to their main connective.
As usual the propositional variable α is *bound* in B, if B contains a subformula of the shape $\forall \alpha.A$. It is *free* otherwise. $\mathrm{FTV}(A)$ denotes the set of free variables of A. Formulae of Soft Linear Logic are considered modulo renaming of bound variables. They are ranged over by A, B, C. The symbol $\equiv$ denotes the syntactical equivalence between formulae modulo renaming of bound variables.
In order to avoid superfluous parentheses, as usual $\multimap$ associates to the right and has precedence over the other connectives. All connectives have precedence over $\forall$, while !

Axiom and cut:

$$\frac{}{A \vdash A}\ (Ax) \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\ (cut)$$

Multiplicatives:

$$\frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\ (\multimap L) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R)$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ (\otimes L) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes R)$$

Multiplicative Unit:

$$\frac{\Gamma \vdash C}{\mathbf{1}, \Gamma \vdash C}\ (\mathbf{1}L) \qquad \frac{}{\vdash \mathbf{1}}\ (\mathbf{1}R)$$

Additives:

$$\frac{\Gamma, A \vdash C}{\Gamma, A \& B \vdash C}\ (\&L_1) \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \& B \vdash C}\ (\&L_2)$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B}\ (\&R)$$

Exponentials:

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

Quantifier:

$$\frac{\Gamma, A[B/\alpha] \vdash C}{\Gamma, \forall \alpha . A \vdash C}\ (\forall L) \qquad \frac{\Gamma \vdash A \quad \alpha \notin \mathrm{FTV}(\Gamma)}{\Gamma \vdash \forall \alpha . A}\ (\forall R)$$

Table 1.1.: Second order Intuitionistic Soft Linear Logic

has precedence on everything else. A^n is a short for $A \otimes \cdots \otimes A$ where A appears n-times. Analogously $!^n A$ is an abbreviation for $!...!A$ where $!$ appears n times. In particular $!^0 A \equiv A$. A formula A is *modal* if $A \equiv !B$ for some formula B. $A[B/\alpha]$ denotes the capture free substitution in A of all occurrences of the propositional variable α by the formula B.
The rules of Soft Linear Logic are depicted in Table 1.1. Soft Linear Logic proves *sequents* of the shape:

$$\Gamma \vdash A$$

where A is an SLL formula and Γ is a *context*. Differently from [Lafont, 2004] where contexts are sequences of SLL formulae, we here consider contexts as *multisets* of Soft Linear Logic formulae. Contexts are denoted by capital greek letters Γ, Δ. The empty multiset is denoted by $\emptyset$ and we usually write $\vdash A$ instead of $\emptyset \vdash A$. The multiset union of Γ and Δ is simply denoted by Γ, Δ. $\mathrm{FTV}(\Gamma)$ denotes the set $\{\alpha \in \mathrm{FTV}(A) \mid A \in \Gamma\}$ while $A^{(n)}$ denotes the multiset $A, \ldots, A$ where A appears $n \geq 0$ times. If $\Gamma = A_1, \ldots, A_n$ then $!\Gamma$ denotes the context $!A_1, \ldots, !A_n$. Proofs in SLL are denoted by Π, Σ, Θ. $\Pi \rhd \Gamma \vdash A$ denotes a proof Π proving the sequent $\Gamma \vdash A$.
The exponential rules:

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

are called respectively *multiplexor* and *soft promotion*, while the natural number n in (m) is the *rank* of the multiplexor rule.
We end this section by recalling a fundamental property of Soft Linear Logic.

Theorem 1. *Soft Linear Logic satisfies cut elimination.*

The reader interested in the proof of the above theorem can consult [Lafont, 2004].

1.2. Some SLL theorems

We consider here some of the logical laws provable in Soft Linear Logic. We detail proofs just to give some examples of SLL derivations. In what follows, we write $A \multimap\!\!\circ\ B$ when both $A \vdash B$ and $B \vdash A$ are provable in SLL.

1.2.1. Linear Theorems

As usual, for SLL we have the following exponential free theorems.

Commutativity $A \otimes B \multimap B \otimes A$, $A \& B \multimap B \& A$, provable by:

$$\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A, B \vdash A \otimes B}\,(\otimes R)}{B \otimes A \vdash A \otimes B}\,(\otimes L) \qquad \text{and} \qquad \dfrac{\dfrac{A \vdash A}{B \& A \vdash A}\,(\&L_2) \quad \dfrac{B \vdash B}{B \& A \vdash A}\,(\&L_1)}{B \& A \vdash A \& B}\,(\&R)$$

Associativity $(A \otimes B) \otimes C \multimap A \otimes (B \otimes C)$, $(A \& B) \& C \multimap A \& (B \& C)$, provable by

$$\dfrac{\dfrac{A \vdash A \quad \dfrac{B \vdash B \quad C \vdash C}{B, C \vdash B \otimes C}\,(\otimes R)}{A, B, C \vdash A \otimes (B \otimes C)}\,(\otimes R)}{A \otimes B, C \vdash A \otimes (B \otimes C)}\,(\otimes L)}{(A \otimes B) \otimes C \vdash A \otimes (B \otimes C)}\,(\otimes L) \quad , \quad \dfrac{\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A, B \vdash A \otimes B}\,(\otimes R) \quad C \vdash C}{A, B, C \vdash (A \otimes B) \otimes C}\,(\otimes R)}{A, B \otimes C \vdash (A \otimes B) \otimes C}\,(\otimes L)}{A \otimes (B \otimes C) \vdash (A \otimes B) \otimes C}\,(\otimes L)$$

$$\dfrac{\dfrac{\dfrac{A \vdash A}{A \& B \vdash A}\,(\&L_1)}{(A \& B) \& C \vdash A}\,(\&L_1) \quad \dfrac{\dfrac{\dfrac{B \vdash B}{A \& B \vdash B}\,(\&L_2)}{(A \& B) \& C \vdash B}\,(\&L_1) \quad \dfrac{C \vdash C}{(A \& B) \& C \vdash C}\,(\&L_2)}{(A \& B) \& C \vdash B \& C}\,(\&R)}{(A \& B) \& C \vdash A \& (B \& C)}\,(\&R)$$

and

$$\dfrac{\dfrac{A \vdash A}{A \& (B \& C) \vdash A}\,(\&L_1) \quad \dfrac{\dfrac{B \vdash B}{B \& C \vdash B}\,(\&L_1)}{A \& (B \& C) \vdash B}\,(\&L_2)}{A \& (B \& C) \vdash A \& B}\,(\&R) \quad \dfrac{\dfrac{C \vdash C}{B \& C \vdash C}\,(\&L_2)}{A \& (B \& C) \vdash C}\,(\&L_2)}{A \& (B \& C) \vdash (A \& B) \& C}\,(\&R)$$

Unit $\mathbf{1} \otimes A \multimap A$, provable by:

$$\dfrac{\dfrac{A \vdash A}{\mathbf{1}, A \vdash A}\,(\mathbf{1}L)}{\mathbf{1} \otimes A \vdash A}\,(\otimes L) \qquad \text{and} \qquad \dfrac{\vdash \mathbf{1} \quad A \vdash A}{A \vdash \mathbf{1} \otimes A}\,(\otimes R)$$

Adjointness $((A \otimes B) \multimap C) \multimap (A \multimap (B \multimap C))$, provable by:

$$\dfrac{\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A, B \vdash A \otimes B}\,(\otimes R) \quad C \vdash C}{A, B, (A \otimes B) \multimap C \vdash C}\,(\multimap L)}{A, (A \otimes B) \multimap C \vdash B \multimap C}\,(\multimap L)}{(A \otimes B) \multimap C \vdash A \multimap (B \multimap C)}\,(\multimap L) \quad \text{and} \quad \dfrac{\dfrac{\dfrac{A \vdash A \quad \dfrac{B \vdash B \quad C \vdash C}{B, B \multimap C \vdash C}\,(\multimap L)}{A, B, A \multimap (B \multimap C) \vdash C}\,(\multimap L)}{A \otimes B, A \multimap (B \multimap C) \vdash C}\,(\otimes L)}{A \multimap (B \multimap C) \vdash (A \otimes B) \multimap C}\,(\multimap R)$$

Distributivity $(A \multimap (B \& C)) \multimap (A \multimap B) \& (A \multimap C)$, provable by:

$$\dfrac{\dfrac{\dfrac{A \vdash A \quad \dfrac{B \vdash B}{B \& C \vdash B}\,(\&L_1)}{A \multimap (B \& C), A \vdash B}\,(\multimap L)}{A \multimap (B \& C) \vdash A \multimap B}\,(\multimap R) \quad \dfrac{\dfrac{A \vdash A \quad \dfrac{C \vdash C}{B \& C \vdash C}\,(\&L_2)}{A \multimap (B \& C), A \vdash C}\,(\multimap L)}{A \multimap (B \& C) \vdash A \multimap C}\,(\multimap R)}{A \multimap (B \& C) \vdash (A \multimap B) \& (A \multimap C)}\,(\&R)$$

and

$$\dfrac{\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A \multimap B, A \vdash B}}{(A \multimap B)\&(A \multimap C), A \vdash B} \quad \dfrac{\dfrac{A \vdash A \quad C \vdash C}{A, A \multimap C \vdash C}\,(\multimap L)}{A, (A \multimap B)\&(A \multimap C) \vdash C}\,(\&L_1)}{\dfrac{(A \multimap B)\&(A \multimap C), A \vdash B\&C}{(A \multimap B)\&(A \multimap C) \vdash A \multimap (B\&C)}}$$

1.2.2. Exponential Theorems

The exponential rules of SLL allows us to prove the following theorems.

Monoidalness 1 $(!A \otimes !B) \multimap !(A \otimes B)$, provable as:

$$\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A \quad B \vdash B}{A, B \vdash A \otimes B}\,(\otimes R)}{!A, !B \vdash !(A \otimes B)}\,(sp)}{!A \otimes !B \vdash !(A \otimes B)}\,(\otimes L)}{\vdash !A \otimes !B \multimap !(A \otimes B)}\,(\multimap R)$$

Monoidalness 2 $!\mathbf{1}$, provable as:

$$\dfrac{\vdash \mathbf{1}}{\vdash !\mathbf{1}}\,(sp)$$

Dereliction $!A \multimap A$ provable as:

$$\dfrac{\dfrac{A \vdash A}{!A \vdash A}\,(m)}{\vdash !A \multimap A}\,(\multimap R)$$

Weakening $!A \multimap \mathbf{1}$, provable as:

$$\dfrac{\dfrac{\vdash \mathbf{1}}{!A \vdash \mathbf{1}}\,(m)}{\vdash !A \multimap \mathbf{1}}\,(\multimap R)$$

Multiplexing $!A \multimap A^n$, provable as:

$$\dfrac{\dfrac{\dfrac{A \vdash A \quad \dfrac{A \vdash A \quad A \vdash A}{A, A \vdash A \otimes A}\,(\otimes R)}{A \vdash A \quad \vdots}\,(\otimes R)}{A^{(n)} \vdash A^n}\,(m)}{!A \vdash A^n}\,(m)}{\vdash !A \multimap A^n}\,(\multimap R)$$

Moreover as usual we have:

Functoriality $A \multimap B$ implies $!A \multimap !B$

1.3. Polynomial time soundness

In this section we recall the complexity properties that make Soft Linear Logic correct for polynomial time computations. We consider the sequent calculus version of SLL introduced at the begin of this section. We refer to [Lafont, 2004] for a proofs of these properties using the proof nets technology.
In the following, we need some measures over SLL proofs.

Definition 2.

- *The* rank *of a proof* Π*, denoted* $\mathtt{rk}(\Pi)$ *is the maximal rank of multiplexor rules in* Π*. A proof* Π *is* homogeneous *if the ranks of the multiplexor rules in* Π *are all the same. A proof* Π *is* generic *if there is no multiplexor in it. A generic proof* Π *will be considered as a homogeneous proof of rank* n *for any* $n \in \mathbb{N}$*.*

- *The* degree *of a proof* Π*, denoted* $\mathtt{d}(\Pi)$ *is the maximal nesting of applications of the* (sp) *rule in* Π*.*

- *Let* r *be a natural number. Then, the* weight $\mathtt{W}(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows:*

  - *If the last applied rule is* (Ax) *or* $(\mathbf{1}R)$ *then* $\mathtt{W}(\Pi, r) = 1$*.*
  - *If the last applied rule is:*

    $$\frac{\Sigma \rhd \Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap R) \qquad \textit{or} \qquad \frac{\Sigma \rhd \Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ (\forall R)$$

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

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

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

    $$\frac{\Sigma \rhd \Gamma \vdash A \quad \Theta \rhd \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes R) \qquad \textit{or} \qquad \frac{\Sigma \rhd \Gamma \vdash A \quad \Theta \rhd \Gamma \vdash B}{\Gamma \vdash A \& B}\ (\& R)$$

    then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r) + 1$*.*
  - *In every other case* $\mathtt{W}(\Pi, r)$ *is the sum of the weights of the premises with respect to* r*.*

Following [Lafont, 2004], we use the word "degree" to denote the maximal nesting of applications of the (sp) rule. The word "depth" is also commonly used in the literature. The next lemma is a direct consequence of the definition of measures.

Lemma 1. *For every* SLL *proof* Π*:*

$$\mathtt{W}(\Pi, \mathtt{rk}(\Pi)) \leq |\Pi|^{\mathtt{d}(\Pi)+1}$$

Here we would show that the weight of a proof strictly decreases when a *cut elimination step* is performed. We consider only *axiom reductions*, reductions of (cut) applications where one of the premise is an axiom, and *symmetric reductions*, reductions of (cut) applications where the left premise ends by a logical right rule and the right premise ends by the corresponding left logical rule. We don't consider *commutative reductions*. In fact the notion of weight has been introduced by Lafont over proof nets, and proof net dispense from considering commutative reductions.

Lemma 2. *Let* Π *and* Π' *be proofs in* SLL*. Then, if* Π' *can be obtained via an external or an axiom reduction by* Π *then:*

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

for every $r \geq \mathtt{rk}(\Pi)$*.*

Proof. We analyze the different cases one by one.

Case $(?)$-(Ax):

$$\dfrac{\dfrac{\Sigma \rhd \Gamma \vdash A \quad \dfrac{}{A \vdash A}\,(Ax)}{\Gamma \vdash A}\,(cut)}{} \quad \text{reduce to} \quad \Sigma \rhd \Gamma \vdash A$$

clearly $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1$ hence conclusion follows trivially. The case (Ax)-$(?)$ is analogous.

Case $(\mathbf{1}R)$-$(\mathbf{1}L)$:

$$\dfrac{\dfrac{}{\vdash \mathbf{1}}\,(\mathbf{1}R) \quad \dfrac{\Sigma \rhd \Gamma \vdash A}{\mathbf{1}, \Gamma \vdash A}\,(\mathbf{1}L)}{\Gamma \vdash B}\,(cut) \quad \text{reduce to} \quad \Sigma \rhd \Gamma \vdash A$$

clearly $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1$ hence conclusion follows trivially.

Case (sp)-(m):

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

reduces to:

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

$\mathtt{W}(\Pi, r) = r\mathtt{W}(\Sigma, r) + 1 + \mathtt{W}(\Theta, r)$ while $\mathtt{W}(\Pi', r) = n\mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r)$, since $r \geq \mathbf{rk}(\Pi) \geq n$ we have: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$.

Case (sp)-(sp):

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

$\mathtt{W}(\Pi, r) = r\mathtt{W}(\Sigma, r) + 1 + r\mathtt{W}(\Theta, r) + 1$ while $\mathtt{W}(\Pi', r) = r(\mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r)) + 1$, hence clearly: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$.

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

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

reduces to:

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

$\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1 + \mathtt{W}(\Theta_1, r) + \mathtt{W}(\Theta_2, r)$ while $\mathtt{W}(\Pi', r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) + \mathtt{W}(\Theta, r)$, hence clearly: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$.

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

$$\dfrac{\dfrac{\Sigma_1 \rhd \Gamma_1 \vdash A \quad \Sigma_2 \rhd \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}\,(\otimes R) \quad \dfrac{\Theta \rhd \Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C}\,(\otimes L)}{\Gamma_1, \Gamma_2, \Delta \vdash C}\,(cut)$$

reduces to:

$$\dfrac{\Sigma_1 \rhd \Gamma_1 \vdash A \qquad \dfrac{\Sigma_2 \rhd \Gamma_2 \vdash B \quad \Theta \rhd \Delta, A, B \vdash C}{\Gamma_2, \Delta, A \vdash C}\,(cut)}{\Gamma_1, \Gamma_2, \Delta \vdash C}\,(cut)$$

$\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) + 1 + \mathtt{W}(\Theta, r)$ while $\mathtt{W}(\Pi', r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) + \mathtt{W}(\Theta, r)$, hence clearly: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$.

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

$$\dfrac{\dfrac{\Sigma_1 \rhd \Gamma \vdash A \quad \Sigma_2 \rhd \Gamma \vdash B}{\Gamma \vdash A \& B}\,(\&R) \quad \dfrac{\Theta \rhd \Delta, A \vdash C}{\Delta, A \& B \vdash C}\,(\&L_1)}{\Gamma, \Delta \vdash C}\,(cut)$$

reduces to:

$$\dfrac{\Sigma_1 \rhd \Gamma \vdash A \quad \Theta \rhd \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\,(cut)$$

$\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) + 1 + \mathtt{W}(\Theta, r)$ while $\mathtt{W}(\Pi', r) = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Theta, r)$, hence clearly: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$.

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

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

reduces to:

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

where $\Sigma[C/\alpha]$ denotes a derivation obtained by Σ substituting every occurrence of α by the formula C, clearly $\mathtt{W}(\Sigma[C/\alpha], r) = \mathtt{W}(\Sigma, r)$.
Then, $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1 + \mathtt{W}(\Theta, r)$ while $\mathtt{W}(\Pi', r) = \mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r)$, hence clearly: $\mathtt{W}(\Pi, r) > \mathtt{W}(\Pi', r)$. □

A direct consequence of the above lemma is the following theorem.

Theorem 2.
A proof Π *reduces, via cut elimination, to normal form in at most* $|\Pi|^{\mathtt{d}(\Pi)+1}$ *steps of axiom and symmetric reductions.*

Proof. By Lemma 1 and Lemma 2. □

The above theorem gives an upper bound on the cut elimination complexity. The complexity becomes polynomial in the size of the proof once the degree is fixed. From a complexity point of view, this is not problematic since, as we will see in the next section, the degree can be considered as a program property.

1.4. PTIME Completeness

The proof that deterministic Turing machines deciding problems in polynomial time in the size of the input can be represented by proofs in Soft Linear Logic is quite involved. Moreover, in the next chapter a similar proof will be detailed. For these reasons, here we only briefly sketch the main expressivity properties that make Soft Linear Logic complete for PTIME.

Firstly we recall that Lafont in [Lafont, 2004] introduces a programming discipline for Soft Linear Logic which consists in representing programs by generic proofs and data by homogeneous proofs.

Let the symbol $\doteq$ denotes definitional equivalence. If $\mathbf{N} \doteq \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$ then natural numbers are definable by homogeneous proofs deriving the sequent:

$$\vdash \mathbf{N}$$

Polynomials in one variable are sufficient to our scope. For technical reasons instead of polynomials Lafont has introduced a notion of *polynomial expressions*, terms built from natural numbers and a variable X, using addition and multiplication. The notation A^n is extended to polynomial expressions:

$$A^X \doteq !A \qquad A^{P+Q} \doteq A^P \otimes A^Q \qquad A^{PQ} \doteq (A^P)^Q$$

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

The following theorem assures that all polynomial expression can be represented in SLL.

Theorem 3. *If P is a polynomial expression, there is a generic proof for the sequent:*

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

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

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

We refer to [Lafont, 2004] for the definition of Turing machine configurations. Here we only recall the PTIME completeness theorem for SLL.

Theorem 4. *If a predicate on boolean strings is computable by a Turing machine in polynomial time $P(n)$ and in polynomial space $Q(n)$, there is a generic proof for the sequent:*

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

which corresponds to this predicate.

2. A Soft Type Assignment system for λ-Calculus

2.1. Introduction

In this chapter we present a type assignment system for λ-calculus inspired by Lafont's Soft Linear Logic [Lafont, 2004]. We start by considering a naive decoration of SLL which lacks the subject reduction property. Here, subject reduction is an important property since we would use the typing to assure the complexity bound on terms reductions. Hence after a finer analysis of the failure of subject reduction in the naive decoration we propose the Soft Type Assignment system (STA).
The idea underlying the introduction of STA is to design a system where terms are built only by substitutions replacing variables occurring once. We show that STA enjoys the subject reduction property and that types can be effectively used to assures complexity properties on λ-terms. Finally we shows that the terms typable in STA are sufficient to define Turing machines working in polynomial time and space.

2.2. Soft Linear Logic and λ-calculus

In this section we recall the naive decoration of Soft Linear Logic presented in [Mairson and Terui, 2003]. This system has the problem of lacking the subject reduction property. In order to better understand this problem we analyze the different ways terms can be built up through applications of rules involving substitutions, in particular applications of the cut rule.

2.2.1. SLL_λ: a naive decoration of SLL

In [Mairson and Terui, 2003] the authors have proposed the system depicted in Table 2.1 as a technical tool for studying the expressive power of SLL. We call such a system SLL_λ. It proves *sequents* of the shape:

$$\Gamma \vdash_{\mathrm{L}} \mathtt{M} : A$$

Axiom and Cut:

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

Multiplicatives:

$$\frac{\Gamma \vdash_{\mathrm{L}} \mathtt{N} : A \quad \mathtt{x} : B, \Delta \vdash_{\mathrm{L}} \mathtt{M} : C \quad \Gamma \# \Delta}{\Gamma, \mathtt{y}^* : A \multimap B, \Delta \vdash_{\mathrm{L}} \mathtt{M}[\mathtt{y}\mathtt{N}/\mathtt{x}] : C}\ (\multimap L) \qquad \frac{\Gamma, \mathtt{x} : A \vdash_{\mathrm{L}} \mathtt{M} : B}{\Gamma \vdash_{\mathrm{L}} \lambda\mathtt{x}.\mathtt{M} : A \multimap B}\ (\multimap R)$$

Exponentials:

$$\frac{\Gamma, \mathtt{x}_1 : A, \ldots, \mathtt{x}_n : A \vdash_{\mathrm{L}} \mathtt{M} : B}{\Gamma, \mathtt{x} : !A \vdash_{\mathrm{L}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : B}\ (m) \qquad \frac{\Gamma \vdash_{\mathrm{L}} \mathtt{M} : A}{!\Gamma \vdash_{\mathrm{L}} \mathtt{M} : !A}\ (sp)$$

Quantifier:

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

(*) y is a fresh variable

Table 2.1.: SLL_λ

where $\mathtt{M}$ is a λ-term, A is a Soft Linear Logic formula and Γ is a context.
In this thesis we will consider only some basic notions about the λ-calculus. The reader interested in a complete treatment can refer to [Barendregt, 1984].
A *context* is a finite set of type assignments of the shape $\mathtt{x} : A$, where $\mathtt{x}$ is a variable and A is a Soft Linear Logic formula. Variables in a context are all distinct. By an abuse of notation, we will denote contexts by Γ, Δ, as for SLL contexts. If $\Gamma = \{\mathtt{x}_1 : A_1, ..., \mathtt{x}_n : A_n\}$, then $\mathrm{dom}(\Gamma) = \{\mathtt{x}_1, ..., \mathtt{x}_n\}$, $\mathrm{rg}(\Gamma) = \{A_1, \dots A_n\}$ and $!\Gamma$ denotes the context $\{\mathtt{x}_1 : !A_1, ..., \mathtt{x}_n : !A_n\}$. $\Gamma \# \Delta$ denotes the fact that $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$. Derivations in SLL_λ are denoted by Π, Σ, Θ. $\Pi \rhd \Gamma \vdash_{\mathrm{L}} \mathtt{M} : \sigma$ denotes a derivation Π with conclusion $\Gamma \vdash_{\mathrm{L}} \mathtt{M} : \sigma$ while $\vdash_{\mathrm{L}} \mathtt{M} : \sigma$ is a short for $\emptyset \vdash_{\mathrm{L}} \mathtt{M} : \sigma$.
Note that the side condition $\Gamma \# \Delta$ in rule $(\multimap L)$ and (cut) is necessary in order to avoid free contractions. In general we will omit it when it is clear from the context.
A fundamental notion in a type assignment system is *typability*.

Definition 3.
A λ-term $\mathtt{M}$ is typable *in SLL_λ if and only if there exists a derivation Π with conclusion $\Gamma \vdash_{\mathrm{L}} \mathtt{M} : A$ for some formula A and some context Γ such that $\mathrm{FV}(\mathtt{M}) \subseteq \mathrm{dom}(\Gamma)$.*

We assume the above notion tacitly extended to other systems presented in the sequel. We can now understand in which sense SLL_λ is a *decoration* of Soft Linear Logic.

Lemma 3.
The sequent $A_1, \dots, A_n \vdash A$ is provable in SLL *if and only if there exist a term $\mathtt{M}$ such that $\mathrm{FV}(\mathtt{M}) \subseteq \{\mathtt{x}_1 \dots, \mathtt{x}_n\}$ and $\mathtt{M}$ is typable in SLL_λ by a derivation with conclusion $\mathtt{x}_1 : A_1, \dots, \mathtt{x}_n : A_n \vdash_{\mathrm{L}} \mathtt{M} : A$.*

Proof. Easy since there is a one-to-one correspondence between Soft Linear Logic rules and SLL_λ rules. □

SLL_λ considered as a type assignment system where types characterize, beside the correctness of terms, also the term reductions complexity, suffers of the failure of an important property: subject reduction. Let us give a detailed analysis of the problem.

2.2.2. The lack of the Subject Reduction Property

In what follows it will be useful to distinguish between three classes of SLL_λ derivations. Consider a derivation Π in SLL_λ with conclusion $\Gamma \vdash_{\mathrm{L}} \mathtt{M} : A$. The formula A can be either modal or not. In the latter case we say that Π is a *linear* derivation. Otherwise we can distinguish two cases. If Π can be transformed by commutations of rules in a

derivation Π' ending by a rule (sp) then we say that Π is *duplicable*. Otherwise we say that it is *shareable*. Note that in other words Π is duplicable if it corresponds to a !-box in the respective proof-net of SLL [Lafont, 2004].
In a decorated sequent calculus system, like SLL_λ, β-reduction is usually considered the counterpart, in terms, of the cut rule of the logic. So consider an occurrence of a rule (cut) as:

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

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

$$\frac{\Pi \rhd \Gamma \vdash_{\mathrm{L}} \mathtt{M} : A \quad \Sigma \rhd \Delta, \mathtt{x} : A \vdash_{\mathrm{L}} \mathtt{N} : B \quad A \text{ not modal} \quad \Pi \text{ linear}}{\Gamma, \Delta \vdash_{\mathrm{L}} \mathtt{N}[\mathtt{M}/\mathtt{x}] : \mathtt{B}}\ (L\ cut)$$

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

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

where L, D and S are shorts for *Linear*, *Duplication* and *Sharing* respectively.
The failure of the subject reduction property is due to the fact that, $(S\ cut)$ can break the correspondence between cut elimination and β-reduction. Let us show it by an example.

Example 1. *Consider the term* $\mathtt{M} \equiv \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})$. *A possible type derivation* Π *for* $\mathtt{M}$ *in* SLL_λ *is:*

$$\frac{\Sigma \rhd \mathtt{s} : A \multimap\, !B, \mathtt{w} : A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{sz})\mathtt{w} : !B \qquad \Theta \rhd \mathtt{y} : B \multimap B \multimap C, \mathtt{x} : !B \vdash_{\mathrm{L}} \mathtt{yxx} : C}{\mathtt{y} : B \multimap B \multimap C, \mathtt{s} : A \multimap\, !B, \mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{yxx}[(\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}/\mathtt{x}] \equiv \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}) : C}\ (S\ cut)$$

where Σ *is the derivation:*

$$\frac{\dfrac{\dfrac{\mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{z} : A \quad \mathtt{r} : !B \vdash_{\mathrm{L}} \mathtt{r} : !B}{\mathtt{s} : A \multimap\, !B, \mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{sz} : !B}\ (\multimap L)}{\mathtt{s} : A \multimap\, !B \vdash_{\mathrm{L}} \lambda \mathtt{z}.\mathtt{sz} : A \multimap\, !B}\ (\multimap R) \quad \dfrac{\mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{w} : A \quad \mathtt{r} : !B \vdash_{\mathrm{L}} \mathtt{r} : !B}{\mathtt{t} : A \multimap\, !B, \mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{tw} : !B}\ (\multimap L)}{\mathtt{s} : A \multimap\, !B, \mathtt{w} : A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{sz})\mathtt{w} : !B}\ (L\ cut)$$

and Θ *is the derivation:*

$$\frac{\mathtt{x}_2 : B \vdash_{\mathrm{L}} \mathtt{x}_2 : B \qquad \dfrac{\mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{x}_1 : B \quad \mathtt{l} : C \vdash_{\mathrm{L}} \mathtt{l} : C}{\mathtt{r} : B \multimap C, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{rx}_1 : C}\ (\multimap L)}{\mathtt{y} : B \multimap B \multimap C, \mathtt{x}_2 : B, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{x}_1 : C}\ (\multimap L)}{\mathtt{y} : B \multimap B \multimap C, \mathtt{x} : !B \vdash_{\mathrm{L}} \mathtt{yxx} : C}\ (m)$$

Note that Σ *is shareable, in fact it cannot be transformed in a derivation ending with* (sp) *rule since there is a non modal formula in the context, i.e.* $A \multimap !B$*. For this reason the last application of a cut rule in* Π *is a* $(S\ cut)$*. For similar reasons the application of a cut rule in* Σ *is a* $(L\ cut)$*.*
Now, since

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

we want to type the reduced term in the same context

$$\mathtt{y} : B \multimap B \multimap C, \mathtt{s} : A \multimap !B, \mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{y}((\mathtt{sw})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}) : C$$

but in fact it is not derivable in SLL_λ*.*

The technical reason is that, in Π, there is a mismatch between the term and the derivation in the sense that in $\mathtt{M}$ there are two copies of $(\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}$, while in Π there is just one subderivation Σ with subject $(\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}$, and this is not duplicable, since in particular $A \multimap !B$ is not modal. So the two copies of $(\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}$ must be treated in a non uniform way.
We want to stress that if instead of using the $(S\ cut)$ we use a $(D\ cut)$, and we modify the types according to this modification, the situation is different.

Example 2. *Consider again the term* $\mathtt{M} \equiv \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})$*. Another derivation* Π_1 *for* $\mathtt{M}$ *in* SLL_λ *is:*

$$\dfrac{\Sigma \rhd \mathtt{s} :!(A \multimap B), \mathtt{w} :!A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{sz})\mathtt{w} :!B \qquad \Theta \rhd \mathtt{y} : B \multimap B \multimap C, \mathtt{x} :!B \vdash_{\mathrm{L}} \mathtt{yxx} : C}{\mathtt{y} : B \multimap B \multimap C, \mathtt{s} :!(A \multimap B), \mathtt{w} :!A \vdash_{\mathrm{L}} \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}) : C}\ (D\ cut)$$

where Σ *is the derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{z} : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{s} : A \multimap B, \mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{sz} : B}\ (\multimap L)}{\mathtt{s} : A \multimap B \vdash_{\mathrm{L}} \lambda \mathtt{z}.\mathtt{sz} : A \multimap B}\ (\multimap R) \quad \dfrac{\mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{w} : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{t} : A \multimap B, \mathtt{w} : A \vdash_{\mathrm{L}} \mathtt{tw} : B}\ (\multimap L)}{\mathtt{s} : A \multimap B, \mathtt{w} : A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{sz})\mathtt{w} : B}\ (L\ cut)}{\mathtt{s} :!(A \multimap B), \mathtt{w} :!A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{sz})\mathtt{w} :!B}\ (sp)$$

and Θ *is the derivation:*

$$\dfrac{\dfrac{\mathtt{x}_2 : B \vdash_{\mathrm{L}} \mathtt{x}_2 : B \qquad \dfrac{\mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{x}_1 : B \quad \mathtt{l} : C \vdash_{\mathrm{L}} \mathtt{l} : C}{\mathtt{r} : B \multimap C, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{rx}_1 : C}\ (\multimap L)}{\mathtt{y} : B \multimap B \multimap C, \mathtt{x}_2 : B, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{x}_1 : C}\ (\multimap L)}{\mathtt{y} : B \multimap B \multimap C, \mathtt{x} :!B \vdash_{\mathrm{L}} \mathtt{yxx} : C}\ (m)$$

Note that Σ *is now duplicable. For this reason the last application of a cut rule in* Π_1 *is a* $(D\ cut)$*. In this case there is no problem in assigning the same type in the*

same context to the reduced term $\mathtt{y(sw)((\lambda z.sz)w)}$. *In fact let* $D \equiv B \multimap B \multimap C$ *and* $E \equiv A \multimap B$. *The term* $\mathtt{y(sw)((\lambda z.sz)w)}$ *can be typed by the following derivation:*

$$\dfrac{\dfrac{\dfrac{\Sigma_2 \rhd \mathtt{s}_2 : E, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{s}_2\mathtt{w}_2 : B \quad \Theta_1 \rhd \mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{yx}_2((\lambda z.\mathtt{s}_1 z)\mathtt{w}_1) : C}{\mathtt{y} : D, \mathtt{s}_1 : E, \mathtt{w}_1 : A, \mathtt{s}_2 : E, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{y}(\mathtt{s}_2\mathtt{w}_2)((\lambda z.\mathtt{s}_1 z)\mathtt{w}_1) : C}\,(m)}{\mathtt{y} : D, \mathtt{s} :! E, \mathtt{w}_1 : A, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{y}(\mathtt{sw}_2)((\lambda z.\mathtt{sz})\mathtt{w}_1) : C}\,(m)}{\mathtt{y} : D, \mathtt{s} :! E, \mathtt{w} :! A \vdash_{\mathrm{L}} \mathtt{y}((\mathtt{sw})((\lambda z.\mathtt{sz})\mathtt{w}) : C}\,(m)$$

where Θ_1 *is the derivation:*

$$\dfrac{\Sigma_1 \rhd \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} (\lambda z.\mathtt{s}_1 z)\mathtt{w}_1 : B \qquad \dfrac{\mathtt{x}_2 : B \vdash_{\mathrm{L}} \mathtt{x}_2 : B \qquad \dfrac{\mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{x}_1 : B \quad \mathtt{l} : C \vdash_{\mathrm{L}} \mathtt{l} : C}{\mathtt{r} : B \multimap C, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{rx}_1 : C}}{\mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{x}_1 : C}\,(L\ cut)}{\mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{yx}_2((\lambda z.\mathtt{s}_1 z)\mathtt{w}_1) : C}\,(L\ cut)$$

Σ_1 *is the derivation:*

$$\dfrac{\dfrac{\dfrac{\mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{z} : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{s}_1 : E, \mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{s}_1\mathtt{z} : B}\,(\multimap L)}{\mathtt{s}_1 : E \vdash_{\mathrm{L}} \lambda z.\mathtt{s}_1\mathtt{z} : E}\,(\multimap R) \qquad \dfrac{\mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{w}_1 : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{t} : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{tw}_1 : B}\,(\multimap L)}{\mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} (\lambda z.\mathtt{s}_1 z)\mathtt{w}_1 : B}\,(L\ cut)$$

and Σ_2 *is the derivation:*

$$\dfrac{\mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{w}_2 : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{s}_2 : E, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{s}_2\mathtt{w}_2 : B}\,(\multimap L)$$

The problem described above is not new, and all the type assignment systems for λ-calculus derived from Linear Logic need to deal with it. Until now the proposed solutions follow three different paths, all based on a natural deduction definition of the type assignment.

The first one, proposed in [Ronchi Della Rocca and Roversi, 1997] based on Intuitionistic Linear Logic, explicitly checks the duplicability condition before to perform a normalization step. So in the resulting language (which is a fully typed λ-calculus) the set of redexes is a proper subset of the set of classical β-redexes.

In [Baillot and Terui, 2004] a type assignment for λ-calculus, based on Light Affine Logic, is designed where the modality ! is no more explicit. In fact there are two arrows, a linear and an intuitionistic one, whose elimination reflects the linear and the duplication cut respectively.

In [Coppola et al., 2005], a type assignment for the λ-calculus, based on Elementary Affine Logic, is designed. There the authors use the call-by-value λ-calculus, where the restricted definition of reduction are reflected exactly in linear and duplication cut.

All the approaches need a careful control of the context, which technically has been realized by splitting it in different parts, collecting respectively the linear and modal assumptions (in case of [Coppola et al., 2005] a further context is needed).
Here we want to explore a different approach. In SLL_λ there is not a direct correspondence between typable terms and derivation structure, in fact the same term can be typed by a plethora of derivations with different structures. Nevertheless, there is a correspondence between such different derivations and the different ways of building the same term. In fact this holds usually for every sequent calculus based type system for λ-calculus, in particular, in the presence of modality.
In SLL_λ the rules (cut) and $(\multimap L)$ build the subject by substitution while the rule (m) can be used to collapse variables. Depending on the application of such rules we have different constructions of the same term.
We call *linear substitution* the substitution acting on variables occurring once in the term. Linear substitutions are at work in the $(L\ cut)$ and they are sufficient for our goals. In fact, a *key observation* is that we can always build a term by linear substitution and by collapsing variables. So we can restrict the set of proofs, in such a way that both sharing and duplication are forbidden, and terms are built by means of linear substitutions only.

Example 3. *Consider again the term* $\mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})$ *and let* $\mathtt{N}_i \equiv (\lambda \mathtt{z}.\mathtt{s}_i\mathtt{z})\mathtt{w}_i$ *for* $i \in \{1,2\}$*,* $D \equiv B \multimap B \multimap C$ *and* $E \equiv A \multimap B$*. It can be typed by the following derivation* Π_2*:*

$$\dfrac{\dfrac{\dfrac{\Sigma_2 \rhd \mathtt{s}_2 : E, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{N}_2 : B \qquad \Theta \rhd \mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{N}_1 : C}{\mathtt{y} : D, \mathtt{s}_1 : E, \mathtt{w}_1 : A, \mathtt{s}_2 : E, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{y}((\lambda \mathtt{z}.\mathtt{s}_2\mathtt{z})\mathtt{w}_2)((\lambda \mathtt{z}.\mathtt{s}_1\mathtt{z})\mathtt{w}_1) \equiv \mathtt{yN}_2\mathtt{N}_1 : C}\,(L\ cut)}{\mathtt{y} : D, \mathtt{s} :! E, \mathtt{w}_1 : A, \mathtt{w}_2 : A \vdash_{\mathrm{L}} \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}_2)((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}_1 : C}\,(m)}{\mathtt{y} : D, \mathtt{s} :! E, \mathtt{w} :! A \vdash_{\mathrm{L}} \mathtt{y}((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda \mathtt{z}.\mathtt{sz})\mathtt{w}) : C}\,(m)$$

where Σ_i *for* $i \in \{1,2\}$ *is the derivation:*

$$\dfrac{\dfrac{\dfrac{\mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{z} : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{s}_i : E, \mathtt{z} : A \vdash_{\mathrm{L}} \mathtt{s}_i\mathtt{z} : B}\,(\multimap L)}{\mathtt{s}_i : E \vdash_{\mathrm{L}} \lambda \mathtt{z}.\mathtt{s}_i\mathtt{z} : E}\,(\multimap R) \qquad \dfrac{\mathtt{w}_i : A \vdash_{\mathrm{L}} \mathtt{w}_i : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{t} : E, \mathtt{w}_i : A \vdash_{\mathrm{L}} \mathtt{tw}_i : B}\,(\multimap L)}{\mathtt{s}_i : E, \mathtt{w}_i : A \vdash_{\mathrm{L}} (\lambda \mathtt{z}.\mathtt{s}_i\mathtt{z})\mathtt{w}_i : B}\,(L\ cut)$$

and Θ *is the derivation:*

$$\dfrac{\Sigma_1 \rhd \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{N}_1 : B \qquad \dfrac{\mathtt{x}_2 : B \vdash_{\mathrm{L}} \mathtt{x}_2 : B \qquad \dfrac{\dfrac{\mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{x}_1 : B \quad \mathtt{l} : C \vdash_{\mathrm{L}} \mathtt{l} : C}{\mathtt{r} : B \multimap C, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{rx}_1 : C}\,(\multimap L)}{\mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{x}_1 : B \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{x}_1 : C}\,(\multimap L)}{\mathtt{y} : D, \mathtt{x}_2 : B, \mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{yx}_2\mathtt{N}_1 : C}\,(L\ cut)$$

Note now that the reduced term $y(sw)((\lambda z.sz)w)$ *can be typed in the same context by replacing* Σ_1 *in* Π_2 *by the following derivation:*

$$\frac{\mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{w}_1 : A \quad \mathtt{r} : B \vdash_{\mathrm{L}} \mathtt{r} : B}{\mathtt{s}_1 : E, \mathtt{w}_1 : A \vdash_{\mathrm{L}} \mathtt{s}_1 w_1 : B} \ (\multimap L)$$

In the Example 1 and Example 2 the term $\mathtt{y}((\lambda\mathtt{z}.\mathtt{sz})\mathtt{w})((\lambda\mathtt{z}.\mathtt{sz})\mathtt{w})$ *has been built by the derivation* Π *and* Π_1 *as:*

$$\mathtt{yx}_2\mathtt{x}_1[\mathtt{x}/\mathtt{x}_1, \mathtt{x}_2][(\lambda\mathtt{z}.\mathtt{sz})\mathtt{w}/\mathtt{x}]$$

while in the Example 3 it has been built by the derivation Π_2 *as:*

$$\mathtt{yx}_1\mathtt{x}_2[(\lambda\mathtt{z}.\mathtt{s}_1\mathtt{z})\mathtt{w}_1/\mathtt{x}_1][(\lambda\mathtt{z}.\mathtt{s}_2\mathtt{z})\mathtt{w}_2/\mathtt{x}_2][\mathtt{s}/\mathtt{s}_1, \mathtt{s}_2][\mathtt{w}/\mathtt{w}_1, \mathtt{w}_2]$$

In order to achieve our goal, we start from an affine version of Soft Linear Logic and restrict the set of types to SLL formulae which are, in some sense that will be made precise in the next section, recursively linear. Such restriction preserves subject reduction since as we will see duplication is a derived rule.

2.3. The Soft Type Assignment System

In this section we present the Soft Type Assignment system (STA). STA assigns to λ-terms as types a proper subset of SLL formulae corresponding in some sense to "recursively linear" formulae where the quantifier can be applied only to linear types. In particular as a consequence of the analysis carried out in the previous section, STA is such that the rules $(\multimap L)$ and (cut) dealing with substitution are applicable only if the type of the replaced variable is linear. So all terms are built through linear substitutions and collapsing of variables.

2.3.1. The system STA

The set of types is a strict subset of Soft Linear Logic formulae.

Definition 4. Linear soft types *and* soft types *are mutually inductively defined as:*

$$A ::= \alpha \mid \sigma \multimap A \mid \forall\alpha.A \quad \textit{(Linear soft types)}$$
$$\sigma ::= A \mid !\sigma$$

The set of soft types *is denoted by* $\mathcal{T}$.

Axiom and Cut:

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

Multiplicatives:

$$\frac{\Gamma \vdash_{\mathrm{T}} \mathtt{N} : \sigma \quad \mathtt{x} : A, \Delta \vdash_{\mathrm{T}} \mathtt{M} : \tau \quad \Gamma \# \Delta}{\Gamma, \mathtt{y}^* : \sigma \multimap A, \Delta \vdash_{\mathrm{T}} \mathtt{M}[\mathtt{y}\mathtt{N}/\mathtt{x}] : \tau}\ (\multimap L) \qquad \frac{\Gamma, \mathtt{x} : \sigma \vdash_{\mathrm{T}} \mathtt{M} : A}{\Gamma \vdash_{\mathrm{T}} \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap R)$$

Exponentials:

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

Quantifier:

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

Weakening:

$$\frac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} : \sigma}\ (w)$$

(*) y is a fresh variable

Table 2.2.: The Soft Type Assignment system.

Type variables are denoted by α, β, linear soft types by A, B, C and soft types by σ, τ, μ. In this chapter we usually omit the prefix "soft" and we simply write linear types and types respectively. As usual the variable α in the type $\forall\alpha.A$ is said *bound* and types will be considered modulo renaming of bound variables. $\sigma[A/\alpha]$ denotes the capture free substitution in σ of all occurrences of the type variable α by the linear type A. $\mathrm{FTV}(\sigma)$ denotes the set of free type variables of σ. As usual $\multimap$ associates to the right and has precedence on $\forall$, while ! has precedence on everything else. In what follows $\forall\vec{\alpha}.A$ is an abbreviation for $\forall\alpha_1....\forall\alpha_m.A$, and $!^n\sigma$ is an abbreviation for $!...!\sigma$ n-times. Note that each type has the shape $!^n\forall\vec{\alpha}.A$.
Terms are considered modulo renaming of bound variables. Variables are ranged over by $\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots$ and λ-terms are ranged over by $\mathtt{M}, \mathtt{N}, \mathtt{P}, \ldots$ As usual, $\mathrm{FV}(\mathtt{M})$ denotes the set of free variables of the term $\mathtt{M}$. $n_o(\mathtt{x}, \mathtt{M})$ is the number of free occurrences of the variable $\mathtt{x}$ in the term $\mathtt{M}$. The symbol $\equiv$ denotes the syntactical equality both for types and terms, modulo renaming of bound variables.
A context Γ is a set of type assignment assumptions of the shape $\mathtt{x} : \sigma$, where all variables are different. By abuse of notation contexts will be ranged over by Γ, Δ. $\mathrm{dom}(\Gamma)$ and $\mathrm{FTV}(\Gamma)$ denotes respectively the set $\{\mathtt{x} \mid \exists \mathtt{x} : \sigma \in \Gamma\}$ and the set $\{\alpha \in \mathrm{FTV}(\sigma) \mid \exists \mathtt{x} : \sigma \in \Gamma\}$. $\Gamma\#\Delta$ denotes the fact that $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$. Let $\Gamma = \mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_n$, then $!\Gamma$ denotes the context $\mathtt{x}_1 :!\sigma_1, \ldots, \mathtt{x}_n :!\sigma_n$.
STA proves sequents of the shape:

$$\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$$

where Γ is a context, $\mathtt{M}$ is a λ-term, and σ is a soft type. The rules defining STA are depicted in Table 2.2. Derivations in STA are denoted by Π, Σ, Θ. $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ denotes a derivation Π with conclusion $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. Moreover, $\vdash_{\mathrm{T}} \mathtt{M} : \sigma$ is a short for $\emptyset \vdash_{\mathrm{T}} \mathtt{M} : \sigma$.
It is worth noting that both the (Ax) and (w) rules can introduce only linear types. The (cut) rule corresponds, according to the classification given in the previous section, to a linear cut. The rule $(\multimap L)$ requires that the type of the replaced variable is linear. These facts and the following lemma assure that all substitutions act on a variable occurring at most once in the subject.

Lemma 4. *$\Gamma, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ implies $n_o(\mathtt{x}, \mathtt{M}) \leq 1$.*

Proof. By induction on the derivation proving $\Gamma, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. The base case is trivial, the other cases follow directly by induction hypothesis remembering the side condition on contexts of (cut) and $(\multimap L)$ rules. □

The following lemma will be useful in the sequel.

Lemma 5. $\Gamma, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} :!\sigma$ *implies* $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.

Proof. By induction on the derivation proving $\Gamma, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} :!\sigma$. The cases where the last applied rule is either (Ax), $(\multimap R)$, $(\forall R)$ or (sp) are not possible. In the case the last applied rule is (w), the conclusion follows trivially. In the cases where the last applied rule is either (cut), $(\forall L)$ or (m), the conclusion follows directly by induction hypothesis. The case where the last applied rule is $(\multimap L)$ follows by induction hypothesis noting that the replaced variable is necessary linear. □

As usual for type systems including second order quantification STA enjoys some substitution properties on types.

Lemma 6.

1. $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ *implies* $\Gamma[\vec{A}/\vec{\alpha}] \vdash_{\mathrm{T}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]$.
2. $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \forall\vec{\alpha}.A$ *implies* $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : A$.

Proof.

1. By induction on the derivation proving $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. The base case is trivial. All the other cases follow directly by induction hypothesis. We show as an example only the case where the last rule is:

$$\frac{\Gamma, \mathtt{x} : B[C/\beta] \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : \forall\beta.B \vdash_{\mathrm{T}} \mathtt{M} : \sigma}\ (\forall L)$$

   By induction hypothesis we have $\Gamma[\vec{A}/\vec{\alpha}], \mathtt{x} : B[C/\beta][\vec{A}/\vec{\alpha}] \vdash_{\mathrm{T}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]$, which can be rewritten as $\Gamma[\vec{A}/\vec{\alpha}], \mathtt{x} : B[\vec{A}/\vec{\alpha}][C[\vec{A}/\vec{\alpha}]/\beta] \vdash_{\mathrm{T}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]$, hence by applying $(\forall L)$ rule we obtain the conclusion:

$$\frac{\Gamma[\vec{A}/\vec{\alpha}], \mathtt{x} : B[\vec{A}/\vec{\alpha}][C[\vec{A}/\vec{\alpha}]/\beta] \vdash_{\mathrm{T}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]}{\Gamma[\vec{A}/\vec{\alpha}], \mathtt{x} : \forall\beta.B[\vec{A}/\vec{\alpha}] \vdash_{\mathrm{T}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]}\ (\forall L)$$

2. By induction on the derivation proving $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \forall\alpha.A$. Let the last rule be:

$$\frac{}{\mathtt{x} : \forall\alpha.A \vdash_{\mathrm{T}} \mathtt{x} : \forall\alpha.A}\ (Ax)$$

   it is easy to build a derivation as

$$\frac{\dfrac{}{\mathtt{x} : A[\alpha/\alpha] \vdash_{\mathrm{T}} \mathtt{x} : A[\alpha/\alpha]}\ (Ax)}{\mathtt{x} : \forall\alpha.A \vdash_{\mathrm{T}} \mathtt{x} : A[\alpha/\alpha]}\ (\forall L)$$

and so the conclusion. The cases where the last rule is $(\multimap R)$ or (sp) are not possible, in the case where the last rule is $(\forall R)$ the conclusion follows directly by the assumption. In every other case the conclusion follows directly by induction hypothesis. □

In what follows we need to identify the positions in a derivation where variables are introduced. In order to do this we define the notion of a chain of $\mathtt{x}$ in $\Pi \rhd \Gamma, \mathtt{x} : \tau \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. A chain will be used to remember the successive renaming of the assumptions which give rise to the assumption $\mathtt{x} : \tau$. The notion of s-chain is necessary to deal with the particular case of the replacement of a variable by another one in a cut rule.

Definition 5.
Let $\Pi \rhd \Gamma, \mathtt{x} : \tau \vdash_{\mathrm{T}} \mathtt{M} : \sigma$.

- *A* chain *of* $\mathtt{x}$ *in* Π *is a sequence of variables inductively defined as follows:*
  - *Let the last applied rule of* Π *be:*

$$\frac{}{\mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{x} : A} \quad , \quad \frac{\Gamma_1 \vdash_{\mathrm{T}} \mathtt{P} : \sigma}{\Gamma_1, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{P} : \sigma} \quad or \quad \frac{\Delta \vdash_{\mathrm{T}} \mathtt{P} : \tau \quad \mathtt{z} : A, \Gamma \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{\mathtt{x} : \tau \multimap A, \Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{xP}/\mathtt{z}] : \sigma}$$

    Then the only chain of $\mathtt{x}$ *is* $\mathtt{x}$ *itself.*
  - *Let the last applied rule of* Π *be:*

$$\frac{\Pi_1 \rhd \Gamma_1, \mathtt{x}_1 : \tau, ..., \mathtt{x}_k : \tau \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{\Gamma_1, \mathtt{x} : !\tau \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_k] : \sigma} \ (m)$$

    Then a chain of $\mathtt{x}$ *in* Π *is every sequence* $\mathtt{x}\vec{\mathtt{c}}$*, where* $\vec{\mathtt{c}}$ *is a chain of* $\mathtt{x}_i$ *in* Π_1 *for* $1 \leq i \leq k$.
  - *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* Σ*. Then a chain of* $\mathtt{x}$ *in* Π *is every sequence* $\mathtt{x}\vec{\mathtt{c}}$*, where* $\vec{\mathtt{c}}$ *is a chain of* $\mathtt{x}$ *in* Σ.
- *Let* $\vec{\mathtt{c}}$ *be a chain of* $\mathtt{x}$ *in* Π *and let* $\mathtt{x}_k$ *be the last variable of* $\vec{\mathtt{c}}$*. Then* $\mathtt{x}_k$ *is an* ancestor *of* $\mathtt{x}$ *in* Π*.* $n_a(\mathtt{x}, \Pi)$ *denotes the number of ancestors of* $\mathtt{x}$ *in* Π.
- $\mathtt{y}$ *is an* effective ancestor *of* $\mathtt{x}$ *if and only if it is an ancestor of* $\mathtt{x}$*, and moreover every variable* $\mathtt{z}$ *in the chain of* $\mathtt{x}$ *ending with* $\mathtt{y}$ *occurs in sequent where* $\mathtt{z}$ *belongs to the free variable set of the subject of the sequent.* $n_e(\mathtt{x}, \Pi)$ *denotes the number of effective ancestors of* $\mathtt{x}$ *in* Π.

- *A* s-chain *of* $\mathtt{x}$ *in* Π *is inductively defined analogously to the chain of* $\mathtt{x}$ *in* Π *but the* (cut) *case:*

$$\frac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : A \quad \Sigma \rhd \Delta, \mathtt{z} : A \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{M}/\mathtt{z}] : \sigma}\ (cut)$$

  If $\mathtt{M} \not\equiv \mathtt{x}$ *then an s-chain of* $\mathtt{x}$ *in* Π *is defined as a chain of* $\mathtt{x}$ *in* Π*, otherwise an s-chain of* $\mathtt{x}$ *in* Π *is every sequence* $\mathtt{z}\vec{\mathtt{c}}$*, where* $\vec{\mathtt{c}}$ *is a chain of* $\mathtt{z}$ *in* Σ*.*
  An s-ancestor *is defined analogously to ancestor, but with respect to s-chains.*

Consider the derivation Π:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{x}_3 : A \vdash_{\mathrm{T}} \mathtt{x}_3 : A}\ (Ax) \quad \dfrac{\dfrac{}{\mathtt{x}_1 : A \vdash_{\mathrm{T}} \mathtt{x}_1 : A}\ (Ax)}{\mathtt{x}_2 : A, \mathtt{x}_1 : A \vdash_{\mathrm{T}} \mathtt{x}_1 : A}\ (w)}{\mathtt{x}_3 : A, \mathtt{x}_2 : A \vdash_{\mathrm{T}} \mathtt{x}_3 : A}\ (cut)}{\mathtt{x} : !A \vdash_{\mathrm{T}} \mathtt{x} : A}\ m$$

Both $\mathtt{x}_3$ and $\mathtt{x}_2$ are ancestors of $\mathtt{x}$ in Π but only $\mathtt{x}_3$ is an effective ancestor. Analogously, $\mathtt{x}_1$ and $\mathtt{x}_2$ are s-ancestors of $\mathtt{x}$ in Π and $\mathtt{x}_1$ is the only effective s-ancestor.
The following lemma follows easily from the previous definition.

Lemma 7. *Let* $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$*. Then* $\forall \mathtt{x} :\ n_o(\mathtt{x}, \mathtt{M}) \leq n_e(\mathtt{x}, \Pi) \leq n_a(\mathtt{x}, \Pi)$*.*

Proof. Easy by induction on Π. □

The notion of effective ancestor will be used in Section 2.4. In what follows, we will need to talk about proofs modulo commutations of rules.

Definition 6. *Let* Π *and* Π' *be two derivations in STA, proving the same conclusion:* $\Pi \rightsquigarrow \Pi'$ *denotes the fact that* Π' *is obtained from* Π *by commuting some rule applications, by erasing* m *rule applications, by inserting* $n \leq m$ *applications of rule* (w)*, for some* $n, m \geq 0$*, and by renaming some variables.*

The Generation Lemma connects the shape of a term with its possible typings, and will be useful in the sequel.
In what follows $\mathtt{M}\{\mathtt{z}\}$ and $\mathtt{M}\{\mathtt{zQ}\}$ denote that $\mathtt{z}$ and $\mathtt{zQ}$ respectively occurs once in $\mathtt{M}$.

Lemma 8 (Generation Lemma).

1. $\Gamma \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{M} : \sigma$ *implies* $\sigma \equiv !^i(\forall \vec{\alpha}.\tau \multimap A)$*, for some* τ*,* A*,* i*,*$\vec{\alpha}$*.*

2. $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{P} : \sigma \multimap A$ *implies* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* $(\multimap R)$*.*

3. $\Pi \rhd \Gamma, \mathtt{x} : \forall\vec{\alpha}.\tau \multimap A \vdash_{\mathrm{T}} \mathtt{M}\{\mathtt{xN}\} : \sigma$ *implies that* Π *is composed by a subderivation* Σ*, followed by a sequence* δ *of rules not containing rule* (sp)*, and the last rule of* Σ *is:*

$$\frac{\Gamma_1 \vdash_{\mathrm{T}} \mathtt{Q} : \tau' \quad \Gamma_2, \mathtt{z} : A' \vdash_{\mathrm{T}} \mathtt{P}\{\mathtt{z}\} : \sigma'}{\Gamma_1, \Gamma_2, \mathtt{y} : \tau' \multimap A' \vdash_{\mathrm{T}} \mathtt{P}\{\mathtt{z}\}[\mathtt{yQ}/\mathtt{z}] : \sigma'} \ (\multimap L)$$

where $\tau' \equiv \tau[\vec{B}/\vec{\alpha}]$*,* $A' \equiv A[\vec{B}/\vec{\alpha}]$*,* $\mathtt{y}$ *is the only s-ancestor of* $\mathtt{x}$ *in* Π*, for some* $\mathtt{P}, \mathtt{Q}, \Gamma_1, \Gamma_2, \vec{B}, \vec{\alpha}, \sigma'$*.*

4. $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{M} : !\sigma$ *implies* $\Pi \rightsquigarrow \Pi'$ *where* Π' *is composed by a subderivation, ending with the rule* (sp) *proving* $!\Gamma' \vdash_{\mathrm{T}} \mathtt{M} : !\sigma$*, followed by a sequence of rules* (w)*,* $(\multimap L)$*,* (m)*,*$(\forall L)$*,* (cut)*, all dealing with variables not occurring in* $\mathtt{M}$*.*

5. $\Pi \rhd !\Gamma \vdash_{\mathrm{T}} \mathtt{M} : !\sigma$ *implies* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* (sp)*.*

6. $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{M} : \forall\alpha.A$ *implies* $\Pi \rightsquigarrow \Pi'$ *where the last rule of* Π' *is* $(\forall R)$*.*

Proof.

1. Easy, by inspection of the rules in Table 2.2.

2. By induction on Π. If the last applied rule is $(\multimap R)$ then the conclusion follows immediately. Otherwise consider the case where $\lambda\mathtt{x}.\mathtt{M} \equiv (\lambda\mathtt{x}.\mathtt{N})[\mathtt{zP}/\mathtt{y}]$ and the last applied rule is:

$$\frac{\Gamma_1 \vdash_{\mathrm{T}} \mathtt{P} : \tau \quad \Sigma \rhd \mathtt{y} : B, \Gamma_2 \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{N} : \sigma \multimap A}{\mathtt{z} : \tau \multimap B, \Gamma_1, \Gamma_2 \vdash_{\mathrm{T}} (\lambda\mathtt{x}.\mathtt{N})[\mathtt{zP}/\mathtt{y}] : \sigma \multimap A} \ (\multimap L)$$

Then by induction Σ can be rewritten as:

$$\frac{\Sigma_1 \rhd \mathtt{y} : B, \Gamma_2, \mathtt{x} : \sigma \vdash_{\mathrm{T}} \mathtt{N} : A}{\mathtt{y} : B, \Gamma_2 \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{N} : \sigma \multimap A} \ (\multimap R)$$

and the desired derivation Π' is:

$$\frac{\dfrac{\Gamma_1 \vdash_{\mathrm{T}} \mathtt{P} : \tau \quad \Sigma_1 \rhd \mathtt{y} : B, \Gamma_2, \mathtt{x} : \sigma \vdash_{\mathrm{T}} \mathtt{N} : A}{\mathtt{z} : \tau \multimap B, \Gamma_1, \Gamma_2, \mathtt{x} : \sigma \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{zP}/\mathtt{y}] : A} \ (\multimap L)}{\mathtt{z} : \tau \multimap B, \Gamma_1, \Gamma_2 \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{N}[\mathtt{zP}/\mathtt{y}] : \sigma \multimap A} \ (\multimap R)$$

The cases where the last applied rule is either (cut), $(\forall L)$ or (m) are similar. The cases where the last applied rule is either (Ax), (sp) or $(\forall R)$ are not possible, while the case where the last applied rule is (w) is trivial.

3. Easy by induction on Π. If the last applied rule is a $(\multimap L)$ rule introducing $\mathtt{x}$ then the conclusion follows immediately. In every other case the conclusion follows by induction hypothesis.

4. By induction on Π. In case the last applied rule is (sp), the proof is obvious. The cases where the last applied rule is either (Ax), $(\multimap R)$ or $(\forall R)$ are not possible. Consider the case $\mathtt{M} \equiv \mathtt{P}[\mathtt{yQ}/\mathtt{x}]$ and the last applied rule is:

$$\frac{\Gamma_1 \vdash_{\mathrm{T}} \mathtt{Q} : \tau \qquad \Sigma \rhd \mathtt{x} : A, \Gamma_2 \vdash_{\mathrm{T}} \mathtt{P} :!\sigma}{\Gamma_1, \Gamma_2, \mathtt{y} : \tau \multimap A \vdash_{\mathrm{T}} \mathtt{P}[\mathtt{yQ}/\mathtt{x}] :!\sigma}\ (\multimap L)$$

   By induction hypothesis $\Sigma \rightsquigarrow \Sigma_1$, where Σ_1 is composed by a subderivation proving $!\Delta \vdash_{\mathrm{T}} \mathtt{P} :!\sigma$, ending with a rule (sp), followed by a sequence δ of rules (w), $(\multimap L)$, (m),$(\forall L)$, (cut), all dealing with variables not occurring in $\mathtt{P}$. Since in the conclusion $\mathtt{x}$ has a linear type, by Lemma 5, it can only be introduced by the sequence δ, so in particular $\mathtt{x} \notin \mathrm{dom}(\Delta)$. $\mathtt{x}$ does not occur in $\mathtt{P}$ consequently $\mathtt{P} \equiv \mathtt{P}[\mathtt{yQ}/\mathtt{x}] \equiv \mathtt{M}$, and the conclusion follows by applying the sequence δ and by some applications of the rule (w).
   The case where the last applied rule is (cut) is similar to the previous one.
   Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n]$ and the last applied rule is:

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

   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 proving $!\Delta_1 \vdash_{\mathrm{T}} \mathtt{N} :!\sigma$, ending with a rule (sp), followed by a sequence δ of rules (w), $(\multimap L)$, (m),$(\forall L)$, (cut), all dealing with variables not occurring in $\mathtt{N}$. Note that for each $\mathtt{x}_i$ with $1 \leq i \leq n$ such that $\mathtt{x}_i \in \mathrm{FV}(\mathtt{N})$ then necessary $\mathtt{x}_i : \tau' \in \Delta_1$ 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:

$$\cfrac{\cfrac{\Delta_2, \mathtt{x}_1 : \tau', \ldots, \mathtt{x}_n : \tau' \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{\Delta_2, \mathtt{x} :!\tau' \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)}{!\Delta_2, \mathtt{x} :!\tau \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] :!\sigma}\ (sp)$$

   followed by a sequence δ_1 of rules (w) recovering the context Δ from the context Δ_2.
   In case the last applied rule is $(\forall L)$, the proof follows directly by induction hypothesis. Also in this case it turn out that $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.

5. By induction on Π. The only possible cases are the ones where the last applied rule is either (sp), (cut) or (m). The case where the last applied rule is (sp) is obvious. Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{P}/\mathtt{x}]$ and the last applied rule is:

$$\frac{!\Delta_1 \vdash_{\mathrm{T}} \mathtt{P} : A \quad \Sigma \rhd !\Delta_2, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{N} :!\sigma}{!\Delta_1, !\Delta_2 \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{P}/\mathtt{x}] :!\sigma}\ (cut)$$

By the point 4 of this lemma $\Sigma \rightsquigarrow \Sigma_1$ where Σ_1 is composed by a subderivation ending as:

$$\frac{\Theta \rhd \Delta_3 \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{!\Delta_3 \vdash_{\mathrm{T}} \mathtt{N} :!\sigma}\ (sp)$$

followed by a sequence δ of rules (w), $(\multimap L)$, (m),$(\forall L)$, (cut), all dealing with variables not occurring in $\mathtt{N}$. Since $\mathtt{x}$ has a linear type then, by Lemma 5, δ needs to contain a rule introducing it. Let δ_1 be the sequence of rules obtained by δ by removing such a rule. Then the desired derivation Π' is Θ, followed by δ_1, followed by (sp).
Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n]$ and the last applied rule is:

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

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

$$\frac{\Theta \rhd \Delta, \mathtt{x}_1 : \tau', ..., \mathtt{x}_n : \tau' \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{!\Delta, \mathtt{x}_1 :!\tau', ..., \mathtt{x}_n :!\tau' \vdash_{\mathrm{T}} \mathtt{N} :!\sigma}\ (sp)$$

So the desired derivation Π' is Θ, followed by a rule (m) and a rule (sp). In the case τ is linear, by Lemma 5, $\mathtt{x}_i \notin \mathrm{FV}(\mathtt{N})$ for each $1 \leq i \leq n$. Moreover by point 4 of this lemma, Σ can be rewritten as:

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

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).

6. By induction on Π. Similar to the proof of point 2 of this lemma. □

2.3.2. Subject reduction

STA enjoys the subject reduction property. The proof is based on the fact that, while formally the (cut) rule is linear, the duplication cut is a derived rule. To prove this, we need a further lemma.

Lemma 9. *$\Pi \rhd \Gamma, \mathtt{x} :!^n \forall\vec{\alpha}.A \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ where $A \not\equiv \forall\vec{\beta}.B$ and $\mathtt{y}$ is an ancestor of $\mathtt{x}$ in Π imply that $\mathtt{y}$ has been introduced with type $\forall\vec{\alpha_1}.A[\vec{B}/\vec{\alpha_2}]$, for some (possible empty) sequences $\vec{\alpha_1}, \vec{\alpha_2}$ such that $\vec{\alpha} = \vec{\alpha_2}, \vec{\alpha_1}$*

Proof. Easy by induction on Π. □

Lemma 10. *The following rule is derivable in STA:*

$$\frac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} :!^n A \quad \mathtt{x} :!^n A, \Delta \vdash_{\mathrm{T}} \mathtt{N} : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{M}/\mathtt{x}] : \sigma}\ (dup)$$

Proof. The case $n = 0$ is trivial, since (dup) coincides with the (cut) rule. So consider the case $n > 0$. Let Π and Σ be derivations with conclusion $\Gamma \vdash_{\mathrm{T}} \mathtt{M} :!^n A$ and $\mathtt{x} :!^n A, \Delta \vdash_{\mathrm{T}} \mathtt{N} : \sigma$ respectively. By Lemma 8.4, Π can be transformed into a derivation Π' composed by a subderivation ending as:

$$\frac{\Pi'' \rhd \Gamma' \vdash_{\mathrm{T}} \mathtt{M} : A}{!^n\Gamma' \vdash_{\mathrm{T}} \mathtt{M} :!^n A}\ (sp)^n$$

followed by a sequence δ of rules dealing with variables not occurring in $\mathtt{M}$. Without loss of generality we can consider $A \equiv \forall\vec{\alpha}.B$. Let the ancestors of $\mathtt{x}$ in Σ be $\mathtt{x}_1, ..., \mathtt{x}_m$. By Lemma 9, for each i such that $1 \leq i \leq m$ the ancestor $\mathtt{x}_i$ has been introduced with type $\forall\vec{\alpha_i'}.A[\vec{C_i}/\vec{\alpha_i''}]$, for some $\vec{C_i}, \vec{\alpha_i'}, \vec{\alpha_i''}$. By Lemma 6, for each i such that $1 \leq i \leq m$ there are disjoint derivations $\Pi_i'' \rhd \Gamma_i' \vdash_{\mathrm{T}} \mathtt{M}_i : \forall\vec{\alpha_i'}.A[\vec{C_i}/\vec{\alpha_i''}]$, where $\mathtt{M}_i$ and Γ_i' are fresh copies of $\mathtt{M}$ and Γ'.
Then, for every ancestor $\mathtt{x}_i$ introduced by an (Ax) rule, replace such rule by Π_i'. Now consider every ancestor $\mathtt{x}_i$ introduced by a $(\multimap L)$ rule as:

$$\frac{\Delta_1 \vdash_{\mathrm{T}} \mathtt{Q} : \tau[\vec{C_i}/\vec{\alpha_i''}] \quad \Delta_2, \mathtt{y} : B'[\vec{C_i}/\vec{\alpha_i''}] \vdash_{\mathrm{T}} \mathtt{P} : \mu}{\Delta_1, \mathtt{x}_i : (\tau \multimap B')[\vec{C_i}/\vec{\alpha_i''}], \Delta_2 \vdash_{\mathrm{T}} \mathtt{P}[\mathtt{x}_i\mathtt{Q}/\mathtt{y}] : \mu}\ (\multimap L)$$

where $\forall\vec{\alpha_i'}.A[\vec{C_i}/\vec{\alpha_i''}] \equiv (\tau \multimap B')[\vec{C_i}/\vec{\alpha_i''}]$ and so $|\vec{\alpha_i'}| = 0$. After this insert the rule:

$$\frac{\Pi_i' \rhd \Gamma_i' \vdash_{\mathrm{T}} \mathtt{M}_i : (\tau \multimap B')[\vec{C_i}/\vec{\alpha_i''}] \quad \Delta_1, \mathtt{x}_i : (\tau \multimap B')[\vec{C_i}/\vec{\alpha_i''}], \Delta_2 \vdash_{\mathrm{T}} \mathtt{P}[\mathtt{x}_i\mathtt{Q}/\mathtt{y}] : \sigma}{\Gamma', \Delta', \Gamma_j \vdash_{\mathrm{T}} \mathtt{P}[\mathtt{M}_i\mathtt{Q}/\mathtt{y}] : \sigma}\ (cut).$$

For every ancestor $\mathtt{x}_i$ introduced by a (w) rule, just erase this rule. Moreover arrange the context and the subject in all rules according with these modifications. Then replace every application of a rule (m), when applied on variables in a chain of $\mathtt{x}$ in Σ, by a sequence of rules (m) applied to the free variables of the corresponding copy of $\mathtt{M}$. So the resulting derivation Θ followed by the sequence δ proves $\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{M}/\mathtt{y}] : \sigma$ and the conclusion follows. □

The rule (dup) doesn't correspond to a $(S\ cut)$, in fact Lemma 8.4 and Lemma 8.5 assure that a derivation with the conclusion proving a modal type corresponds to a !-box.
The above lemma allow us to freely use (dup) rule in what follows.

Theorem 5 (Subject Reduction). *If* $\Gamma \vdash_T M : \sigma$ *and* $M \to_\beta M'$ *then* $\Gamma \vdash_T M' : \sigma$

Proof. By induction on the derivation $\Theta \rhd \Gamma \vdash_T M : \sigma$. The only interesting case is when the last rule is (cut), creating the redex $(\lambda y.P)Q$ reduced in M. So, without loss of generality, consider the case $M \equiv N\{xQ\}[\lambda y.P/x]$ and the last applied rule is:

$$\frac{\Pi \rhd \Gamma \vdash_T \lambda y.P : A \quad \Sigma \rhd \Delta, x : A \vdash_T N\{xQ\} : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash_T N\{xQ\}[\lambda y.P/x] : \sigma}\ (cut).$$

By Lemma 8.1, and by the constraint on the (cut) rule, $A \equiv \forall \vec{\alpha}.\tau \multimap B$. By Lemma 8.3, Σ is composed by a subderivation Σ_1, followed by a sequence δ of rules not containing (sp). Σ_1 ends as:

$$\frac{\Theta_1 \rhd \Delta_1 \vdash_T Q' : \tau' \quad \Theta_2 \rhd \Delta_2, z : B' \vdash_T N'\{z\} : \sigma'}{\Delta_1, x' : \tau' \multimap B', \Delta_2 \vdash_T N'\{z\}[x'Q'/z] : \sigma'}$$

where $\tau' \equiv \tau[\vec{C}/\vec{\alpha}]$, $B' \equiv B[\vec{C}/\vec{\alpha}]$ and x' is the only s-ancestor of x, for some $\Delta_1, \Delta_2, N', Q', \sigma', \vec{C}$. By Lemma 8.2 and Lemma 8.6, $\Pi \rightsquigarrow \Pi_1$ where Π_1 ends as:

$$\frac{\dfrac{\Gamma, y : \tau \vdash_T P : B}{\Gamma \vdash_T \lambda y.P : \tau \multimap B}\ (\multimap R)}{\Gamma \vdash_T \lambda y.P : \forall \vec{\alpha}.\tau \multimap B}\ (\forall R)^*$$

So by Lemma 6.1, there is a derivation $\Pi_2 \rhd \Gamma, y : \tau' \vdash_T P : B'$, since $\vec{\alpha}$ are not free in Γ. Without loss of generalities $\tau' \equiv !^n A'$ for some linear type A'. So we can apply the derived rule (dup) obtaining a derivation Θ_3 ending as:

$$\frac{\Theta_1 \rhd \Delta_1 \vdash_T Q' :!^n A' \quad \Pi_2 \rhd \Gamma, y :!^n A' \vdash_T P : B'}{\Gamma, \Delta_1 \vdash_T P[Q'/y] : B'}\ (dup)$$

Hence we can conclude:

$$\frac{\Theta_3 \rhd \Gamma, \Delta_1 \vdash_T P[Q'/y] : B' \quad \Theta_2 \rhd \Delta_2, z : B' \vdash_T N'\{z\} : \sigma'}{\Gamma, \Delta_1, \Delta_2 \vdash_T N'\{z\}[P[Q'/y]/z] : \sigma}\ (cut)$$

and by applying the sequence of rules δ the desired derivation Θ' can be built. □

The proof of subject reduction described above sets the correspondences between the β-reduction mechanism and the cut elimination procedure. In particular β-reduction corresponds to the usual *small-step* cut elimination procedure (local transformation of the derivation) in case the bound variable has a linear type (there is no need of duplication). Instead it corresponds to a *big-step* cut elimination procedure (global transformation of the derivation in the style of natural deduction normalization) in case

the bound variable has a modal type (a duplication of the argument is necessary).
A similar big-step cut elimination procedure, named q-protocol, has been described in [Danos et al., 1997, Danos and Joinet, 2003] for classical logic and elementary linear logic sequent calculus respectively.
Example 4 shows how the big-step cut elimination procedure works on the term $(\lambda x.yxx)((\lambda z.sz)w)$.

2.4. Complexity

In this section we show that STA is correct and complete for polynomial time complexity. We prove that, if a term $\mathtt{M}$ can be typed in STA by a derivation Π, then it reduces to normal form in a number of β-reduction steps which is bounded by $|\mathtt{M}|^{\mathtt{d}(\Pi)+1}$, where $|\mathtt{M}|$ is the number of symbols of $\mathtt{M}$ and $\mathtt{d}(\Pi)$ is the number of nested applications of rule (sp) in Π. So working with terms typed by derivations of fixed degree assures us to keep only a polynomial number of computation steps. Moreover, since the typing gives also a polynomial upper bound on the size of terms that can be obtained by reduction, the polynomial time soundness follows.
Moreover, we prove that STA is complete both for PTIME and FPTIME, using a representation of Turing machine working in polynomial time by λ-terms, typable in STA through derivations obeying suitable constraints.
The idea behind this kind of characterization is that data can be represented by terms typable by linear types using derivations of degree 0. We allow programs to duplicate their data. Then, a derivation typing an application of a program to its data, in general, can have degree greater than 0, but this degree depends only on the program. So, the complexity measure is well formulated.

2.4.1. Complexity of reductions in STA

Definition 7. *Let* Π *be a derivation in* STA. *Then for every subderivation* Π' *of* Π *ending as:*

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{N} : A \quad \Delta, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma}\ (cut) \quad or \quad \frac{\Sigma \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{N} : \mu \quad \Delta, \mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{y} : \mu \multimap A, \Delta \vdash_{\mathrm{T}} \mathtt{M}[\mathtt{y}\mathtt{N}/\mathtt{x}] : \sigma}\ (\multimap L)$$

the subderivation Σ *of* Π' *is* erasing *if* $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.

SLL enjoys a polynomial time bound, namely the cut-elimination procedure is polynomial in the size of the proof. This result holds obviously also for STA. But we need something more, namely to relate the polynomial bound to the size of the λ-terms. So

Example 4. *Consider the following derivation* Π *for the term* $(\lambda x.yxx)((\lambda z.sz)w)$*:*

$$\dfrac{\dfrac{\dfrac{}{x_2 : A \vdash_T x_2 : A} \quad \dfrac{\dfrac{}{x_1 : A \vdash_T x_1 : A} \quad \dfrac{}{t : A \vdash_T t : A}}{r : A \multimap A, x_1 : A \vdash_T rx_1 : A}}{y : A \multimap A \multimap A, x_1 : A, x_2 : A \vdash_T yx_2x_1 : A}}{\dfrac{y : A \multimap A \multimap A, x :!A \vdash_T yxx : A}{y : A \multimap A \multimap A \vdash_T \lambda x.yxx :!A \multimap A}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{z : A \vdash_T z : A} \quad \dfrac{}{t : A \vdash_T t : A}}{s : A \multimap A, z : A \vdash_T sz : A}}{s : A \multimap A \vdash_T \lambda z.sz : A \multimap A} \quad \dfrac{\dfrac{}{w : A \vdash_T w : A} \quad \dfrac{}{t : A \vdash_T t : A}}{r : A \multimap A, w : A \vdash_T rw : A}}{\dfrac{s : A \multimap A, w : A \vdash_T (\lambda z.sz)w : A}{s :!(A \multimap A), w :!A \vdash_T (\lambda z.sz)w :!A}} \quad \dfrac{}{z : A \vdash_T z : A}}{t :!A \multimap A, s :!(A \multimap A), w :!A \vdash_T t((\lambda z.sz)w) : A}$$

$$y : A \multimap A \multimap A, s :!(A \multimap A), w :!A \vdash_T (\lambda x.yxx)((\lambda z.sz)w) : A$$

since $(\lambda x.yxx)((\lambda z.sz)w) \to_\beta y((\lambda z.sz)w)((\lambda z.sz)w)$ *the big-step cut elimination procedure transform* Π *in the following derivation* Π'*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : A \vdash_T z : A} \quad \dfrac{}{t : A \vdash_T t : A}}{s_1 : A \multimap A, z : A \vdash_T s_1z : A}}{s_1 : A \multimap A \vdash_T \lambda z.s_1z : A \multimap A} \quad \dfrac{\dfrac{}{w_1 : A \vdash_T w_1 : A} \quad \dfrac{}{t : A \vdash_T t : A}}{r : A \multimap A, w_1 : A \vdash_T rw_1 : A}}{s_1 : A \multimap A, w_1 : A \vdash_T (\lambda z.s_1z)w_1 : A} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{z : A \vdash_T z : A} \quad \dfrac{}{t : A \vdash_T t : A}}{s_2 : A \multimap A, z : A \vdash_T s_2z : A}}{s_2 : A \multimap A \vdash_T \lambda z.s_2z : A \multimap A} \quad \dfrac{\dfrac{}{w_2 : A \vdash_T w_2 : A} \quad \dfrac{}{t : A \vdash_T t : A}}{r : A \multimap A, w_2 : A \vdash_T rw_2 : A}}{s_2 : A \multimap A, w_2 : A \vdash_T (\lambda z.s_2z)w_2 : A} \quad \dfrac{}{r : A \vdash_T r : A}}{t : A \multimap A, s_2 : A \multimap A, w_2 : A \vdash_T t((\lambda z.s_2z)w_2) : A}}{y : A \multimap A \multimap A, s_1 : A \multimap A, s_2 : A \multimap A, w_1 : A, w_2 : A \vdash_T y((\lambda z.s_1z)w_1)((\lambda z.s_2z)w_2) : A}}{y : A \multimap A \multimap A, s :!(A \multimap A), w_1 : A, w_2 : A \vdash_T y((\lambda z.sz)w_1)((\lambda z.sz)w_2) : A}}{y : A \multimap A \multimap A, s :!(A \multimap A), w :!A \vdash_T y((\lambda z.sz)w)((\lambda z.sz)w) : A}$$

we prove this result by defining notions of measures of both terms and proofs. Such measures are an adaptation of that given by Lafont, where the subderivations that do not contribute to the term formation, which we have called above "erasing", are not considered.

Definition 8.

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

- *The* rank *of a rule* (m) *as:*

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

  is the number $k \leq n$ *of variables* $\mathtt{x}_i$ *such that* $\mathtt{x}_i \in \mathrm{FV}(\mathtt{M})$ *for* $1 \leq i \leq n$.
  Let r *be the the maximum rank of a rule* (m) *in* Π*, not considering erasing subderivations. The rank* $\mathtt{rk}(\Pi)$ *of* Π *is the maximum between* 1 *and* r.

- *The* degree $\mathtt{d}(\Pi)$ *of* Π *is the maximum nesting of applications of rule* (sp) *in* Π*, not considering erasing subderivations;*

- *Let* r *be a natural number. Then, the* weight $\mathtt{W}(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows:*

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

$$\frac{\Sigma \rhd \Gamma, \mathtt{x} : \sigma \vdash_{\mathrm{T}} \mathtt{M} : A}{\Gamma \vdash_{\mathrm{T}} \lambda\mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap R)$$

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

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

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

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

    then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + \mathtt{W}(\Theta, r) - 1$ *if* $\mathtt{x} \in \mathrm{FV}(\mathtt{M})$, $\mathtt{W}(\Pi, r) = \mathtt{W}(\Theta, r)$ *otherwise.*

– *In every other case* $\mathtt{W}(\Pi, r)$ *is the sum of the weights of the premises with respect to* r*, not counting erasing subderivations.*

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

Lemma 11.
Let $\Pi \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$*. Then:*

1. $\mathtt{rk}(\Pi) \leq |\mathtt{M}| \leq |\Pi|$.
2. $\mathtt{W}(\Pi, r) \leq r^{\mathrm{d}(\Pi)}\mathtt{W}(\Pi, 1)$
3. $\mathtt{W}(\Pi, 1) = |\mathtt{M}|$.
4. $\mathtt{x} :!^q A \in \Gamma$ *implies* $n_o(\mathtt{x}, \mathtt{M}) \leq n_e(\mathtt{x}, \Pi) \leq \mathtt{rk}(\Pi)^q$.
5. $\Pi \rightsquigarrow \Pi'$ *implies* $\mathtt{W}(\Pi', r) \leq \mathtt{W}(\Pi, r)$.

Proof.

1. The first inequality follows directly by the definition of rank. The second one holds because the constraints on the rules (cut) and $(\multimap L)$ assure that terms are linearly built.
2. Easy by induction on Π. Note that by definition the weight is multiplied by r only by instances of the (sp) rule, so the exponent corresponds to the degree of the proof.
3. By induction on Π. The base case is trivial. Consider the case where Π ends as:

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

   By induction hypothesis $\mathtt{W}(\Sigma, 1) = |\mathtt{N}|$ and $\mathtt{W}(\Theta, 1) = |\mathtt{M}|$. If $\mathtt{x} \in \mathrm{FV}(\mathtt{M})$ then by Lemma 4 $|\mathtt{M}[\mathtt{N}/\mathtt{x}]| = |\mathtt{M}| + |\mathtt{N}| - 1 = \mathtt{W}(\Theta, 1) + \mathtt{W}(\Sigma, 1) - 1$ hence the conclusion follows. Otherwise $|\mathtt{M}[\mathtt{N}/\mathtt{x}]| = |\mathtt{M}| = \mathtt{W}(\Theta, 1)$ and also in this case the conclusion follows.
   Consider the case where Π ends as:

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{N} : \mu \quad \Theta \rhd \mathtt{x} : A, \Delta \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{y} : \mu \multimap A, \Delta \vdash_{\mathrm{T}} \mathtt{M}[\mathtt{y}\mathtt{N}/\mathtt{x}] : \sigma}\ (\multimap L)$$

   By induction hypothesis $\mathtt{W}(\Sigma, 1) = |\mathtt{N}|$ and $\mathtt{W}(\Theta, 1) = |\mathtt{M}|$. If $\mathtt{x} \in \mathrm{FV}(\mathtt{M})$ then clearly $|\mathtt{M}[\mathtt{y}\mathtt{N}/\mathtt{x}]| = |\mathtt{M}| + |\mathtt{N}| = \mathtt{W}(\Theta, 1) + \mathtt{W}(\Sigma, 1)$ hence the conclusion follows. Otherwise

$|\mathtt{M}[\mathtt{yN}/\mathtt{x}]| = |\mathtt{M}| = \mathtt{W}(\Theta, 1)$ and also in this case the conclusion follows since Σ is erasing.
The other cases follows directly by induction hypothesis remembering that erasing subderivations must not be counted.

4. The first inequality holds because the constraints on the rules (cut) and $(\multimap L)$ assure that terms are linearly built and in particular that variable are linearly introduced.
The second inequality can be easily proved by induction using Lemma 4 for the base case.

5. The erasing of rules can decrease the weight while the commutation of rules and the insertion of (w) rules leave the weight unchanged. □

The following lemma extends the weight definition to the derived rule (dup) by taking into account the weight of the involved proofs.

Lemma 12. *Let Θ be a derivation ending with the derived rule:*

$$\frac{\Pi \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{M} :!^n A \quad \Sigma \rhd \mathtt{x} :!^n A, \Delta \vdash_{\mathrm{T}} \mathtt{N} : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{M}/\mathtt{x}] : \sigma}\ (dup)$$

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

$$\mathtt{W}(\Theta, r) \leq \mathtt{W}(\Pi, r) + \mathtt{W}(\Sigma, r)$$

Proof. The proof is analogous to the proof of Lemma 10 but taking in consideration how the weights are modified.
The case $n = 0$ is trivial, since (dup) coincides with the (cut) rule. So consider the case $n > 0$. By Lemma 8.5, Π can be transformed into a derivation Π' composed by a subderivation ending as:

$$\frac{\Pi'' \rhd \Gamma' \vdash_{\mathrm{T}} \mathtt{M} : A}{!^n\Gamma' \vdash_{\mathrm{T}} \mathtt{M} :!^n A}\ (sp)^n$$

followed by a sequence δ of rules dealing with variables not occurring in $\mathtt{M}$. So by Lemma 11.6 and by definition of weight since δ deals only with variable not occurring in $\mathtt{M}$: $\mathtt{W}(\Pi, r) = r^n\mathtt{W}(\Pi'', r)$. In particular for each fresh copy Π''_i of Π'' clearly $\mathtt{W}(\Pi''_i, r) = \mathtt{W}(\Pi'', r)$.
Now, consider $\sigma \equiv !^k B$. By Lemma 8.4 and Lemma 8.5 Σ can be transformed in a derivation Σ' followed by $0 \leq k \leq n$ applications of rule (sp) and by a sequence of rules dealing with variables not occurring in $\mathtt{N}$. So in particular $\mathtt{W}(\Sigma, r) = r^k\mathtt{W}(\Sigma', r)$.

Then, for every ancestor $\mathtt{x}_i$ in Σ the replacement by Π_i'' increases the weight $\mathtt{W}(\Sigma', r)$ of at most a quantity $\mathtt{W}(\Pi', r)$. By definition of weight we are interested only in effective ancestors, hence by Lemma 11.4:

$$\begin{aligned}\mathtt{W}(\Theta, r) &= r^k(\mathtt{W}(\Sigma', r) + n_e(\mathtt{x}, \Sigma')\mathtt{W}(\Pi'', r)) \\ &\leq r^k\mathtt{W}(\Sigma', r) + r^k r^{n-k}\mathtt{W}(\Pi'', r) \\ &\leq r^k\mathtt{W}(\Sigma', r) + r^n\mathtt{W}(\Pi'', r) \\ &= \mathtt{W}(\Sigma, r) + \mathtt{W}(\Pi, r)\end{aligned}$$

□

Now we can show that the weight decreases when a β-reduction is performed.

Lemma 13. *Let $\Theta \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M} : \sigma$ and $\mathtt{M} \to_\beta \mathtt{M}'$. There is a derivation $\Theta' \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M}' : \sigma$, with $\mathtt{rk}(\Theta) \geq \mathtt{rk}(\Theta')$, such that if $r \geq \mathtt{rk}(\Theta')$ then $\mathtt{W}(\Theta', r) < \mathtt{W}(\Theta, r)$.*

Proof. The proof is analogous to the proof of Theorem 5 but taking in consideration how the weight are modified, using Lemma 11.
We will use exactly the same notations as in the theorem. Note that a sequence of rule δ not containing rule (sp) increment a weight of a constant factor c which depends on δ. So looking at the definition of weight and to the theorem we can state the following (in)equalities:

$$\begin{aligned}\mathtt{W}(\Theta', r) &= \mathtt{W}(\Theta_3, r) + \mathtt{W}(\Theta_2, r) + c - 1 \\ &\leq \mathtt{W}(\Theta_1, r) + \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Theta_2, r) + c - 1 \\ &= \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Sigma_1, r) + c - 1 \\ &= \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Sigma, r) - 1 \\ &< \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Sigma, r) - 1 \\ &= \mathtt{W}(\Theta, r)\end{aligned}$$

□

Finally the desired results can be obtained.

Theorem 6 (Strong Polystep Soundness). *Let $\Pi \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M} : \sigma$, and $\mathtt{M}$ β-reduces to $\mathtt{M}'$ in m steps. Then:*

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

Proof.

1. By Lemma 11.2, Lemma 11.3 and by repeatedly using Lemma 13, since $|\mathtt{M}| \geq \mathtt{rk}(\Pi)$.
2. By repeatedly using Lemma 13 there is a derivation $\Sigma \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M}' : \sigma$ such that $\mathtt{W}(\Sigma, r) < \mathtt{W}(\Pi, r)$. By Lemma 11.3 $|\mathtt{M}'| = \mathtt{W}(\Sigma, 1)$. Since clearly $\mathtt{W}(\Sigma, 1) \leq \mathtt{W}(\Sigma, r)$ we have $|\mathtt{M}'| < \mathtt{W}(\Pi, r)$ and by Lemma 11.2 the conclusion follows. □

Theorem 7 (Polytime Soundness). *Let $\Pi \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M} : \sigma$, then $\mathtt{M}$ can be evaluated to normal form on a Turing machine in time $O(|\mathtt{M}|^{3(\mathtt{d}(\Pi)+1)})$.*

Proof. Clearly, as pointed in [Terui, 2001], a β reduction step $\mathtt{N} \to_\beta \mathtt{N}'$ can be simulated in time $O(|\mathtt{N}|^2)$ on a Turing machine. Let $\mathtt{M} \equiv \mathtt{M}_0 \to_\beta \mathtt{M}_1 \to_\beta \cdots \to_\beta \mathtt{M}_n$ be a reduction of $\mathtt{M}$ to normal form $\mathtt{M}_n$. By Theorem 6.2 $|\mathtt{M}_i| \leq |\mathtt{M}|^{\mathtt{d}(\Pi)+1}$ for $0 \leq i \leq n$, hence each step in the reduction takes time $O(|\mathtt{M}|^{2(\mathtt{d}(\Pi)+1)})$. Furthermore since by Theorem 6.1 n is $O(|\mathtt{M}|^{\mathtt{d}(\Pi)+1})$, the conclusion follows. □

Theorem 7 holds for every strategy. In fact analogously to [Terui, 2002] it could have been formulated as a strong polytime soundness, considering Turing machine with an oracle for strategies. We refer to [Terui, 2002] for further details.

2.4.2. Polynomial Time Completeness

In order to prove polynomial time completeness for STA we need to encode Turing machines (TM) configurations, transitions between such configurations and iterators. We encode input data in the usual way, TM configurations and transitions following the lines of [Mairson and Terui, 2003] and iterators as usual by Church numerals.
We stress that we allow a liberal typing of terms. In fact we don't request an analogous of Lafont's programming discipline, i.e. the distinction between programs as terms typable by generic proofs and data as terms typable by homogeneous proofs.
Nevertheless, in order to make the bound in Theorem 6 polynomial, we show that each input data can be typed through derivations with fixed degree. In particular we show that they are typable with derivation with degree equal to 0.

Definability We generalize the usual notion of lambda definability, given in [Barendregt, 1984], to different kinds of input data.

Definition 9. *Let $f : \mathbb{I}_1 \times \cdots \times \mathbb{I}_n \to \mathbb{O}$ be a total function and let elements $o \in \mathbb{O}$ and $i_j \in \mathbb{I}_j$, for $0 \leq j \leq n$, be encoded by terms $\underline{\mathtt{o}}$ and $\underline{\mathtt{i}_j}$ such that $\Delta \vdash_{\mathtt{T}} \underline{\mathtt{o}} : \mathbf{O}$ and $\Delta_j \vdash_{\mathtt{T}} \underline{\mathtt{i}_j} : \mathbf{I}_j$. Then, f is* definable *if, there exists a term $\underline{\mathtt{f}} \in \Lambda$ such that $\Delta \vdash_{\mathtt{T}} \underline{\mathtt{f}}\underline{\mathtt{i}_1} \cdots \underline{\mathtt{i}_n} : \mathbf{O}$ and:*

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

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

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

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

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

Tensor product Tensor product is definable by second order quantifier as

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

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

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

The following are derived rules:

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

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

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

In what follows σ^n denotes $\sigma \otimes \cdots \otimes \sigma$ n-times. Note that, since STA is an affine system, tensor product enjoys some properties of the additive conjunction, as to allow the projectors.

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

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

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

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

The following are derived rules:

$$\frac{}{\vdash_{\mathrm{T}} \mathtt{I} : \mathbf{1}} \qquad \frac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : \mathbf{1} \vdash_{\mathrm{T}} \mathtt{let\ z\ be\ I\ in\ M} : \sigma}$$

Church numerals To encode natural numbers we will use Church numerals:

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

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

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

nevertheless we prefer, for reasons that will become clear, to introduce the following more general notion of indexed type.

Definition 10. *The* indexed type $\mathbf{N}_i$ *for each* $i \in \mathbb{N}$ *is defined as:*

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

Clearly $\mathbf{N}_1 \equiv \mathbf{N}$. We have introduced indexed types since they give more information on the typing of terms than the usual ones. An example of this is the following lemma.

Lemma 14. *For each Church numerals* $\underline{\mathtt{n}}$ *and for each* $i > 0 \in \mathbb{N}$*:*

$$\vdash_{\mathrm{T}} \underline{\mathtt{n}} : \mathbf{N}_i$$

Proof. By induction on n. We show only an example by considering how to type the church numeral $\underline{2}$ with a generic index $i > 0$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{z} : \alpha}\,(Ax) \quad \dfrac{}{\mathtt{l} : \alpha \vdash_{\mathrm{T}} \mathtt{l} : \alpha}\,(Ax)}{\mathtt{s}_2 : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_2\mathtt{z} : \alpha}\,(\multimap L) \quad \dfrac{}{\mathtt{l} : \alpha \vdash_{\mathrm{T}} \mathtt{l} : \alpha}\,(Ax)}{\mathtt{s}_1 : \alpha \multimap \alpha, \mathtt{s}_2 : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_1(\mathtt{s}_2\mathtt{z}) : \alpha}\,(\multimap L)}{\mathtt{s} :!^i(\alpha \multimap \alpha), \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}(\mathtt{sz}) : \alpha}\,(m)^i}{\mathtt{s} :!^i(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \lambda \mathtt{z}.\mathtt{s}(\mathtt{sz}) : \alpha \multimap \alpha}\,(\multimap R)}{\vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{sz}) :!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha}\,(\multimap R)}{\vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{sz}) : \forall \alpha.!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha}\,(\forall R)$$

□

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

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

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

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

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

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

It is worth noting that in contrast with what happens in linear logic here the step function is iterable only if it is definable through a term typable with type $A \multimap A$ for some linear type A.

Polynomials Successor, addition and multiplication are definable through the usual terms [Barendregt, 1984].

Lemma 15. *The term* $\underline{\mathtt{succ}} \doteq \lambda \mathtt{p}.\lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{psz})$ *defines the successor function and is typable in* STA *as* $\vdash_{\mathrm{T}} \underline{\mathtt{succ}} : \mathbf{N}_i \multimap \mathbf{N}_{i+1}$ *for each* $i > 0 \in \mathbb{N}$.

Proof. It is well known that $\underline{\mathtt{succ}}$ behaves on Church numerals as a successor, so we only need to show that it is typable in STA. This can be proved by induction on i using the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{s}_2 : \alpha \multimap \alpha \vdash_{\mathrm{T}} \mathtt{s}_2 : \alpha \multimap \alpha}}{\mathtt{s}_2 :!^i(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{s}_2 :!^i(\alpha \multimap \alpha)}\,(sp)^i \quad \dfrac{\dfrac{\dfrac{}{\mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{z} : \alpha} \quad \dfrac{}{\mathtt{l} : \alpha \vdash_{\mathrm{T}} \mathtt{l} : \alpha}}{\mathtt{x} : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{xz} : \alpha} \quad \dfrac{}{\mathtt{l} : \alpha \vdash_{\mathrm{T}} \mathtt{l} : \alpha}}{\mathtt{s}_1 : \alpha \multimap \alpha, \mathtt{x} : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_1(\mathtt{xz}) : \alpha}}{\mathtt{p} :!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha, \mathtt{s}_2 :!^i(\alpha \multimap \alpha), \mathtt{s}_1 : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_1(\mathtt{ps}_2\mathtt{z}) : \alpha}}{\mathtt{p} : \mathbf{N}_i, \mathtt{s}_2 :!^i(\alpha \multimap \alpha), \mathtt{s}_1 : \alpha \multimap \alpha, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_1(\mathtt{ps}_2\mathtt{z}) : \alpha}}{\mathtt{p} : \mathbf{N}_i, \mathtt{s}_2 :!^i(\alpha \multimap \alpha), \mathtt{s}_1 :!^i(\alpha \multimap \alpha), \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}_1(\mathtt{ps}_2\mathtt{z}) : \alpha}\,(m)^i}{\mathtt{p} : \mathbf{N}_i, \mathtt{s} :!^{i+1}(\alpha \multimap \alpha), \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{s}(\mathtt{psz}) : \alpha}\,(m)}{\mathtt{p} : \mathbf{N}_i, \mathtt{s} :!^{i+1}(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \lambda \mathtt{z}.\mathtt{s}(\mathtt{psz}) : \alpha \multimap \alpha}}{\mathtt{p} : \mathbf{N}_i \vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{psz}) :!^{i+1}(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha}}{\mathtt{p} : \mathbf{N}_i \vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{psz}) : \mathbf{N}_{i+1}}}{\vdash_{\mathrm{T}} \lambda \mathtt{p}.\lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{psz}) : \mathbf{N}_i \multimap \mathbf{N}_{i+1}}$$

□

Lemma 16. *The term* $\underline{\mathtt{add}} \doteq \lambda \mathtt{p}.\lambda \mathtt{q}.\lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{ps}(\mathtt{qsz})$ *defines the addition function and is typable in* STA *as* $\vdash_{\mathrm{T}} \underline{\mathtt{add}} : \mathbf{N}_i \multimap \mathbf{N}_j \multimap \mathbf{N}_{\max(i,j)+1}$ *for each* $i, j > 0 \in \mathbb{N}$.

Proof. It is easy to verify that $\underline{\mathtt{add}}$ behaves on Church numerals as the addition function, so we only need to show that it is typable in STA. This can be proved by induction on

i and j using the following derivation where $A = \alpha \multimap \alpha$.

$$\dfrac{\mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{z} : \alpha \qquad \dfrac{\mathtt{m} : \alpha \vdash_{\mathrm{T}} \mathtt{m} : \alpha \quad \mathtt{r} : \alpha \vdash_{\mathrm{T}} \mathtt{r} : \alpha}{\mathtt{m} : \alpha, \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{lm} : \alpha}}{\mathtt{z} : \alpha, \mathtt{r} : A, \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{l(rz)} : \alpha}$$

$$\dfrac{\mathtt{s}_2 : A \vdash_{\mathrm{T}} \mathtt{s}_2 : A}{\mathtt{s}_2 :!^j A \vdash_{\mathrm{T}} \mathtt{s}_2 :!^j A}\,(sp)^j \qquad \dfrac{\mathtt{s}_1 : A \vdash_{\mathrm{T}} \mathtt{s}_1 : A}{\mathtt{s}_1 :!^i A \vdash_{\mathrm{T}} \mathtt{s}_1 :!^i A}\,(sp)^i$$

$$\dfrac{\mathtt{s}_2 :!^j A \vdash_{\mathrm{T}} \mathtt{s}_2 :!^j A \qquad \mathtt{z} : \alpha, \mathtt{r} : A, \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{l(rz)} : \alpha}{\mathtt{q} :!^j A \multimap A, \mathtt{s}_2 :!^j A, \mathtt{l} : A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{l(qs_2z)} : \alpha}$$

$$\dfrac{\mathtt{s}_1 :!^i A \vdash_{\mathrm{T}} \mathtt{s}_1 :!^i A \qquad \mathtt{q} :!^j A \multimap A, \mathtt{s}_2 :!^j A, \mathtt{l} : A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{l(qs_2z)} : \alpha}{\dfrac{\dfrac{\dfrac{\mathtt{p} :!^i A \multimap A, \mathtt{q} :!^j A \multimap A, \mathtt{s}_1 :!^i A, \mathtt{s}_2 :!^j A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps_1(qs_2z)} : \alpha}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} :!^y A \multimap A, \mathtt{s}_1 :!^i A, \mathtt{s}_2 :!^j A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps_1(qs_2z)} : \alpha}}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j, \mathtt{s}_1 :!^i A, \mathtt{s}_2 :!^j A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps_1(qs_2z)} : \alpha}}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j, \mathtt{s}_1 :!^{\max(i,j)} A, \mathtt{s}_2 :!^{\max(i,j)} A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps_1(qs_2z)} : \alpha}\,(m)^{[\max(i,j)-\min(i,j)]}}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j, \mathtt{s}_1 :!^{\max(i,j)} A, \mathtt{s}_2 :!^{\max(i,j)} A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps_1(qs_2z)} : \alpha}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j, \mathtt{s} :!^{\max(i,j)+1} A, \mathtt{z} : \alpha \vdash_{\mathrm{T}} \mathtt{ps(qsz)} : \alpha}}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j, \mathtt{s} :!^{\max(i,j)+1} A \vdash_{\mathrm{T}} \lambda \mathtt{z.ps(qsz)} : A}}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j \vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z.ps(qsz)} :!^{\max(i,j)+1} A \multimap A}}{\mathtt{p} : \mathbf{N}_i, \mathtt{q} : \mathbf{N}_j \vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{z.ps(qsz)} : \mathbf{N}_{\max(i,j)+1}}}{\mathtt{p} : \mathbf{N}_i \vdash_{\mathrm{T}} \lambda \mathtt{q}.\lambda \mathtt{s}.\lambda \mathtt{z.ps(qsz)} : \mathbf{N}_j \multimap \mathbf{N}_{\max(i,j)+1}}}{\vdash_{\mathrm{T}} \lambda \mathtt{p}.\lambda \mathtt{q}.\lambda \mathtt{s}.\lambda \mathtt{z.ps(qsz)} : \mathbf{N}_i \multimap \mathbf{N}_j \multimap \mathbf{N}_{\max(i,j)+1}}$$

□

Lemma 17. *The term* $\underline{\mathtt{mul}} \doteq \lambda \mathtt{p}.\lambda \mathtt{q}.\lambda \mathtt{s.p(qs)}$ *defines the multiplication function and is typable in* STA *as* $\vdash_{\mathrm{T}} \underline{\mathtt{mul}} : \mathbf{N}_j \multimap !^j \mathbf{N}_i \multimap \mathbf{N}_{i+j}$ *for each* $i, j > 0 \in \mathbb{N}$.

Proof. It is easy to verify that $\underline{\mathtt{mul}}$ behaves on Church numerals as the multiplication function, so we only need to show that it is typable in STA. This can be proved by induction on i and j using the following derivation.

$$\dfrac{\dfrac{\dfrac{\mathtt{s} : \alpha \multimap \alpha \vdash_{\mathrm{T}} \mathtt{s} : \alpha \multimap \alpha}{\mathtt{s} :!^i(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{s} :!^i(\alpha \multimap \alpha)}\,(sp)^i \qquad \mathtt{m} : \alpha \multimap \alpha \vdash_{\mathrm{T}} \mathtt{m} : \alpha \multimap \alpha}{\mathtt{p} :!^i(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha, \mathtt{s} :!^i(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{ps} : \alpha \multimap \alpha}}{\dfrac{\mathtt{p} : \mathbf{N}_i, \mathtt{s} :!^i(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{ps} : \alpha \multimap \alpha}{\mathtt{p} :!^j \mathbf{N}_i, \mathtt{s} :!^{i+j}(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{ps} :!^j(\alpha \multimap \alpha)}\,(sp)^j}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{p} :!^j \mathbf{N}_i, \mathtt{s} :!^{i+j}(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{ps} :!^j(\alpha \multimap \alpha) \qquad \mathtt{m} : \alpha \multimap \alpha \vdash_{\mathrm{T}} \mathtt{m}}{\mathtt{q} :!^j(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha, \mathtt{p} :!^j \mathbf{N}_i, \mathtt{s} :!^{i+j}(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{q(ps)} : \alpha \multimap \alpha}}{\mathtt{q} : \mathbf{N}_j, \mathtt{p} :!^j \mathbf{N}_i, \mathtt{s} :!^{i+j}(\alpha \multimap \alpha) \vdash_{\mathrm{T}} \mathtt{q(ps)} : \alpha \multimap \alpha}}{\mathtt{q} : \mathbf{N}_j, \mathtt{p} :!^j \mathbf{N}_i \vdash_{\mathrm{T}} \lambda \mathtt{s.q(ps)} :!^{i+j}(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha}}{\mathtt{q} : \mathbf{N}_j, \mathtt{p} :!^j \mathbf{N}_i \vdash_{\mathrm{T}} \lambda \mathtt{s.q(ps)} : \mathbf{N}_{i+j}}}{\mathtt{q} : \mathbf{N}_j \vdash_{\mathrm{T}} \lambda \mathtt{p}.\lambda \mathtt{s.q(ps)} :!^j \mathbf{N}_i \multimap \mathbf{N}_{i+j}}}{\vdash_{\mathrm{T}} \lambda \mathtt{q}.\lambda \mathtt{p}.\lambda \mathtt{s.q(ps)} : \mathbf{N}_j \multimap !^j \mathbf{N}_i \multimap \mathbf{N}_{i+j}}$$

□

Note that the terms $\underline{\mathtt{succ}}$, $\underline{\mathtt{add}}$ and $\underline{\mathtt{mul}}$ cannot be typed using only the type $\mathbf{N}$, in fact this is the reason why we have introduced the notion of indexed types. We have chosen to introduce indexed types instead of using a translation of STA into itself as done by Lafont in the completeness proof for SLL because it is a more natural typing for such usual λ-terms. In fact for addition and multiplication Lafont in [Lafont, 2004] has used proof-nets with underlying λ-terms different from the usual one.
It should also be remarked that the above terms cannot be iterated as step function through Church numerals iteration, indeed they can be typed only in a non uniform way, i.e. using a type $A \multimap B$ with $A \not\equiv B$.
Even if the terms $\underline{\mathtt{succ}}$, $\underline{\mathtt{add}}$ and $\underline{\mathtt{mul}}$ cannot be iterated, they can be composed to obtain all the polynomials. Note that also the composition is not uniform.

Example 5. *If we want to multiply two natural numbers we can use* $\underline{\mathtt{mul}}$ *typed as:*

$$\vdash_{\mathrm{T}} \underline{\mathtt{mul}} : \mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash_{\mathrm{T}} \underline{\mathtt{mul}} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3 \quad \Pi \rhd \mathtt{r} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N}, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{r}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : \mathbf{N}_3}{\mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N}, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \underline{\mathtt{mul}}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : \mathbf{N}_3}}{\mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \lambda \mathtt{z}.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : !!\mathbf{N} \multimap \mathbf{N}_3}}{\mathtt{x} : \mathbf{N} \vdash_{\mathrm{T}} \lambda \mathtt{y}.\lambda \mathtt{z}.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : !\mathbf{N} \multimap !!\mathbf{N} \multimap \mathbf{N}_3}}{\vdash_{\mathrm{T}} \lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z}.\underline{\mathtt{mul}}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : \mathbf{N} \multimap !\mathbf{N} \multimap !!\mathbf{N} \multimap \mathbf{N}_3}$$

where Π *is the following derivation:*

$$\dfrac{\Theta \rhd \mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y} : \mathbf{N}_2 \quad \Sigma \rhd \mathtt{r} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{t} : \mathbf{N}_2, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{rtz} : \mathbf{N}_3}{\mathtt{r} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N}, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{r}(\underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y})\mathtt{z} : \mathbf{N}_3}$$

Θ *is the derivation:*

$$\dfrac{\vdash_{\mathrm{T}} \underline{\mathtt{mul}} : \mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2 \qquad \dfrac{\mathtt{x} : \mathbf{N} \vdash_{\mathrm{T}} \mathtt{x} : \mathbf{N} \qquad \dfrac{\dfrac{\mathtt{y} : \mathbf{N} \vdash_{\mathrm{T}} \mathtt{y} : \mathbf{N}}{\mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \mathtt{y} : !\mathbf{N}} \quad \mathtt{o} : \mathbf{N}_2 \vdash_{\mathrm{T}} \mathtt{o} : \mathbf{N}_2}{\mathtt{p} : !\mathbf{N} \multimap \mathbf{N}_2, \mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \mathtt{py} : \mathbf{N}_2}}{\mathtt{q} : \mathbf{N} \multimap !\mathbf{N} \multimap \mathbf{N}_2, \mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \mathtt{qxy} : \mathbf{N}_2}}{\mathtt{x} : \mathbf{N}, \mathtt{y} : !\mathbf{N} \vdash_{\mathrm{T}} \underline{\mathtt{mul}}\,\mathtt{x}\,\mathtt{y} : \mathbf{N}_2}$$

and Σ *is the derivation:*

$$\dfrac{\dfrac{\mathtt{t} : \mathbf{N}_2 \vdash_{\mathrm{T}} \mathtt{t} : \mathbf{N}_2 \quad \mathtt{u} : !!\mathbf{N} \multimap \mathbf{N}_3 \vdash_{\mathrm{T}} \mathtt{u} : !!\mathbf{N} \multimap \mathbf{N}_3}{\mathtt{r} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{t} : \mathbf{N}_2 \vdash_{\mathrm{T}} \mathtt{rt} : !!\mathbf{N} \multimap \mathbf{N}_3} \quad \dfrac{\dfrac{\mathtt{z} : \mathbf{N} \vdash_{\mathrm{T}} \mathtt{z} : \mathbf{N}}{\mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{z} : !!\mathbf{N}} \quad \mathtt{l} : \mathbf{N}_3 \vdash_{\mathrm{T}} \mathtt{l} : \mathbf{N}_3}{\mathtt{s} : !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{sz} : \mathbf{N}_3}}{\mathtt{r} : \mathbf{N}_2 \multimap !!\mathbf{N} \multimap \mathbf{N}_3, \mathtt{t} : \mathbf{N}_2, \mathtt{z} : !!\mathbf{N} \vdash_{\mathrm{T}} \mathtt{rtz} : \mathbf{N}_3}$$

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

It is important to stress that the change of type as in the Example 5, in particular on the number of modalities, does not depend on the values of data.
We can finally show that STA is complete for polynomials.

Lemma 18. *Let* P *be a polynomial in the variable* X *and* $deg(P)$ *its degree. Then there is a term* $\underline{\mathtt{P}}$ *defining* P *typable as :*

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

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

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

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

$$\mathtt{x}_1 : \mathbf{N}_i, \ldots, \mathtt{x}_n :!^{i(deg(P')-1)}\mathbf{N}_i \vdash_{\mathrm{T}} \underline{\mathtt{P}'} : \mathbf{N}_{i(deg(P'))+deg(P')+1}$$

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

$$\mathtt{x}_0 : \mathbf{N}_i, \mathtt{x}_1 :!^i\mathbf{N}_i, \ldots, \mathtt{x}_n :!^{i(deg(P')-1)+i}\mathbf{N}_i \vdash_{\mathrm{T}} \underline{\mathtt{P}^*} : \mathbf{N}_{i(deg(P')+1)+deg(P')+1+1}$$

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

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

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

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

Booleans Let us encode booleans, as usual, by:

$$\mathbf{0} \doteq \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x} \qquad \mathbf{1} \doteq \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y} \qquad \mathtt{if\ x\ then\ M\ else\ N} \doteq \mathtt{xMN}$$

By convention we use $\mathbf{0}$ and $\mathbf{1}$ for *true* and *false* respectively. They can be typed in STA by the usual linear logic type for booleans:

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

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

$$\frac{\mathtt{b} \in \{\mathbf{0},\mathbf{1}\}}{\Gamma \vdash_{\mathrm{T}} \mathtt{b} : \mathbf{B}}\ (\mathbf{B}R) \qquad \frac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma \quad \Delta \vdash_{\mathrm{T}} \mathtt{N} : \sigma \quad \Gamma \# \Delta}{\Gamma, \Delta, \mathtt{x} : \mathbf{B} \vdash_{\mathrm{T}} \ \mathtt{if\ x\ then\ M\ else\ N} : \sigma}\ (\mathbf{B}L)$$

With the help of the conditional we can define all the usual boolean functions

Lemma 19.
The terms $\underline{\mathtt{And}} \doteq \lambda \mathtt{b}_1.\lambda \mathtt{b}_2.\mathtt{if\ b}_1\ \mathtt{then\ b}_2\ \mathtt{else}\ \mathbf{1}$, $\underline{\mathtt{Or}} \doteq \lambda \mathtt{b}_1.\lambda \mathtt{b}_2.\mathtt{if\ b}_1\ \mathtt{then}\ \mathbf{1}\ \mathtt{else\ b}_2$ *and* $\underline{\mathtt{Not}} \doteq \lambda \mathtt{b}_1.\mathtt{if\ b}_1\ \mathtt{then}\ \mathbf{1}\ \mathtt{else}\ \mathbf{0}$ *define the boolean conjunction, disjunction and negation respectively. They are typable as:*

$$\vdash_{\mathrm{T}} \underline{\mathtt{And}} : \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B} \qquad \vdash_{\mathrm{T}} \underline{\mathtt{Or}} : \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B} \qquad \vdash_{\mathrm{T}} \underline{\mathtt{Not}} : \mathbf{B} \multimap \mathbf{B}$$

Proof. Easy by using the derived rules $(\mathbf{B}L)$ and $(\mathbf{B}R)$. □

Furthermore we have two terms defining boolean contraction and weakening respectively.

Lemma 20.
The terms $\underline{\mathtt{Cnt}} \doteq \lambda \mathtt{b}.\ \mathtt{if\ b\ then}\ \langle \mathbf{0},\mathbf{0}\rangle\ \mathtt{else}\ \langle \mathbf{1},\mathbf{1}\rangle$ *and* $\underline{\mathtt{Weak}} \doteq \lambda \mathtt{b}.\lambda \mathtt{x}.\mathtt{x}$ *define the boolean contraction and weakening respectively. They are typable as:*

$$\vdash_{\mathrm{T}} \underline{\mathtt{Cnt}} :: \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B} \qquad \vdash_{\mathrm{T}} \underline{\mathtt{Weak}} : \mathbf{B} \multimap \mathbf{1}$$

Proof. Easy. □

The above functions are useful to prove the following lemma.

Lemma 21. *Each boolean total function* $f : \mathbb{B}^n \to \mathbb{B}^m$*, where* $n, m \geq 1$*, can be defined by a term* $\underline{\mathtt{f}}$ *typable in* STA *as* $\vdash_{\mathrm{T}} \underline{\mathtt{f}} : \mathbf{B}^n \multimap \mathbf{B}^m$*.*

Proof. It follows easily by Lemma 20 and Lemma 19. □

Strings String of booleans can be encoded as:

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

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

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

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

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

In what follows we will use the following.

Lemma 22. *The term* $\mathbf{len} \doteq \lambda \mathsf{c}.\lambda \mathsf{s}.\mathsf{c}(\lambda \mathsf{x}.\lambda \mathsf{y}.\mathsf{sy})$ *defines the function returning the length of an input string. It is typable in* STA *with typing:*

$$\vdash_{\mathrm{T}} \mathbf{len} : \mathbf{S}_i \multimap \mathbf{N}_i$$

Proof. Easy. □

Turing Machine We can define Turing machine configurations by terms of the shape

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

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

Definition 11. *The* indexed type $\mathbf{TM}_i^q$ *for each* $i, q \in \mathbb{N}$ *is defined as:*

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

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

Lemma 23. *Let* $n \in \mathbb{N}$. *Every term* $\lambda \mathsf{c}.\langle \mathsf{cb}_0^l \circ \cdots \circ \mathsf{cb}_n^l, \mathsf{cb}_0^r \circ \cdots \circ \mathsf{cb}_m^r, \langle \mathsf{q}_0, \ldots, \mathsf{q}_n \rangle \rangle$ *defines a Turing machine configuration. For every* $i > 0$ *such terms are typable in* STA *as:*

$$\vdash_{\mathrm{T}} \lambda \mathsf{c}.\langle \mathsf{cb}_0^l \circ \cdots \circ \mathsf{cb}_n^l, \mathsf{cb}_0^r \circ \cdots \circ \mathsf{cb}_m^r, \langle \mathsf{q}_0, \ldots, \mathsf{q}_n \rangle \rangle : \mathbf{TM}_i^n$$

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

Lemma 24. *The term* $\mathbf{Init} \doteq \lambda \mathsf{t}.\lambda \mathsf{c}.\langle \lambda \mathsf{z}.\mathsf{z}, \lambda \mathsf{z}.\mathsf{t}(\mathsf{c}\mathbf{0})\mathsf{z}, \underline{\mathsf{Q}_0} \rangle$ *defines the function that, taking as input a Church numeral* $\underline{\mathsf{n}}$*, gives as output a Turing machine with tape of length* n *filled by* $\mathbf{0}$*'s in the initial state* $\mathsf{Q}_0 \equiv \langle \mathsf{q}_0, \ldots, \mathsf{q}_n \rangle$ *and with the head at the beginning of the tape. For each* $i \in \mathbf{N}$ *it is typable in* STA *as:*

$$\vdash_{\mathrm{T}} \mathbf{Init} : \mathbf{N}_i \multimap \mathbf{TM}_i^n$$

Proof. Easy. □

Following [Mairson and Terui, 2003], in order to show that Turing machine transitions are definable we consider two distinct phases. In the first one the TM configuration is decomposed to extract the first symbol of each part of the tape. In the second phase the symbols obtained in the previous one are combined, depending on the transition function, to reconstruct the tape after the transition step. In order to type the decomposition of a TM configuration we will use the type $\mathbf{ID}_i$ defined as:

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

For the decomposition phase we have the following lemma.

Lemma 25. *The term:*

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

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

$$\vdash_{\mathrm{T}} \mathbf{Dec} : \mathbf{TM}_i \multimap \mathbf{ID}_i$$

Its behaviour is to decompose a configuration as:

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

Proof. In order to type $\mathbf{Dec}$ it is convenient to have some type abbreviations:

$$\begin{array}{lcllcl} A & = & \alpha \multimap \alpha & \qquad D & = & (C \multimap C) \otimes (C \multimap C) \otimes \mathbf{B}^q \\ B & = & \mathbf{B} \multimap \alpha \multimap \alpha & \qquad E & = & A \otimes A \otimes B \otimes \mathbf{B} \otimes B \otimes \mathbf{B} \otimes \mathbf{B}^q \\ C & = & A \otimes B \otimes \mathbf{B} & & & \end{array}$$

and some term abbreviations:

$$\begin{array}{lcl} \mathtt{P} & = & \langle \mathtt{t}_l, \mathtt{t}_r, \mathtt{c}_l, \mathtt{b}_0^l, \mathtt{c}_r, \mathtt{b}_0^r, \mathtt{q}\rangle \\ \mathtt{Q} & = & \mathtt{let\ x}_l\ \mathtt{be\ t}_l, \mathtt{c}_l, \mathtt{b}_0^l\ \mathtt{in\ let\ x}_r\ \mathtt{be\ t}_r, \mathtt{c}_r, \mathtt{b}_0^r\ \mathtt{in\ P} \\ \mathtt{N} & = & \langle \mathtt{I}, \lambda\mathtt{x}.\mathtt{I}, \mathbf{0}\rangle \\ \mathtt{L} & = & \mathtt{let\ lN\ be\ t}_l, \mathtt{c}_l, \mathtt{b}_0^l\ \mathtt{in\ let\ x}_r\ \mathtt{be\ t}_r, \mathtt{c}_r, \mathtt{b}_0^r\ \mathtt{in\ P} \\ \mathtt{M} & = & \mathtt{let\ lN\ be\ t}_l, \mathtt{c}_l, \mathtt{b}_0^l\ \mathtt{in\ let\ rN\ be\ t}_r, \mathtt{c}_r, \mathtt{b}_0^r\ \mathtt{in\ P} \end{array}$$

$\mathtt{F}[\mathtt{c}]$ is typable in STA by the following derivation Π:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{i}:\mathbf{B}\vdash_{\mathrm{T}}\mathtt{i}:\mathbf{B}}\quad\dfrac{}{\mathtt{s}:A\vdash_{\mathrm{T}}\mathtt{s}:A}}{\mathtt{h}:B,\mathtt{i}:\mathbf{B}\vdash_{\mathrm{T}}\mathtt{hi}:A}\qquad\dfrac{}{\mathtt{g}:A\vdash_{\mathrm{T}}\mathtt{g}:A}}{\mathtt{h}:B,\mathtt{i}:\mathbf{B},\mathtt{g}:A\vdash_{\mathrm{T}}\mathtt{hi}\circ\mathtt{g}:A}\qquad\dfrac{}{\mathtt{c}:B\vdash_{\mathrm{T}}\mathtt{c}:B}}{\mathtt{h}:B,\mathtt{i}:\mathbf{B},\mathtt{g}:A,\mathtt{c}:B\vdash_{\mathrm{T}}\langle\mathtt{hi}\circ\mathtt{g},\mathtt{c}\rangle:A\otimes B}\qquad\dfrac{}{\mathtt{b}:\mathbf{B}\vdash_{\mathrm{T}}\mathtt{b}:\mathbf{B}}}{\mathtt{h}:B,\mathtt{i}:\mathbf{B},\mathtt{g}:A,\mathtt{c}:B,\mathtt{b}:\mathbf{B}\vdash_{\mathrm{T}}\langle\mathtt{hi}\circ\mathtt{g},\mathtt{c},\mathtt{b}\rangle:C}}{\mathtt{z}:C,\mathtt{b}:\mathbf{B},\mathtt{c}:B\vdash_{\mathrm{T}}\mathtt{let\ z\ be\ g,h,i\ in}\ \langle\mathtt{hi}\circ\mathtt{g},\mathtt{c},\mathtt{b}\rangle:C}}{\mathtt{b}:\mathbf{B},\mathtt{c}:B\vdash_{\mathrm{T}}\lambda\mathtt{z}.\mathtt{let\ z\ be\ g,h,i\ in}\ \langle\mathtt{hi}\circ\mathtt{g},\mathtt{c},\mathtt{b}\rangle:C\multimap C}}{\mathtt{c}:B\vdash_{\mathrm{T}}\lambda\mathtt{b}.\lambda\mathtt{z}.\mathtt{let\ z\ be\ g,h,i\ in}\ \langle\mathtt{hi}\circ\mathtt{g},\mathtt{c},\mathtt{b}\rangle:\mathbf{B}\multimap C\multimap C}$$

It is quite obvious how to construct a derivation $\Theta\rhd\mathtt{t}_l:A,\mathtt{t}_r:A,\mathtt{c}_l:B,\mathtt{b}_0^l:\mathbf{B},\mathtt{c}_r:B,\mathtt{b}_0^r:\mathbf{B},\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{P}:E$ so, let Θ_1 be the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{t}_l:A,\mathtt{t}_r:A,\mathtt{c}_l:B,\mathtt{b}_0^l:\mathbf{B},\mathtt{c}_r:B,\mathtt{b}_0^r:\mathbf{B},\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{P}:E}}{\mathtt{x}_r:C,\mathtt{t}_l:A,\mathtt{c}_l:B,\mathtt{b}_0^l:\mathbf{B},\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{let\ x}_r\ \mathtt{be\ t}_r,\mathtt{c}_r,\mathtt{b}_0^r\ \mathtt{in\ P}:E}}{\mathtt{x}_l:C,\mathtt{x}_r:C,\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{Q}:E}$$

and Σ be the following:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{x}:\alpha\vdash_{\mathrm{T}}\mathtt{x}:\alpha}}{\vdash_{\mathrm{T}}\mathtt{I}:\alpha\multimap\alpha}}{\mathtt{x}:\mathbf{B}\vdash_{\mathrm{T}}\mathtt{I}:\alpha\multimap\alpha}\qquad\dfrac{}{\vdash_{\mathrm{T}}\mathtt{0}:\mathbf{B}}}{\vdash_{\mathrm{T}}\langle\mathtt{I},\lambda\mathtt{x}.\mathtt{I},\mathtt{0}\rangle:((\alpha\multimap\alpha)\otimes(\mathbf{B}\multimap\alpha\multimap\alpha)\otimes\mathbf{B})}$$

Hence we have a derivation Σ_1 as:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma\rhd\vdash_{\mathrm{T}}\mathtt{N}:C}\qquad\dfrac{\dfrac{}{\Sigma\rhd\vdash_{\mathrm{T}}\mathtt{N}:C}\qquad\dfrac{}{\Theta_1\rhd\mathtt{x}_l:C,\mathtt{x}_r:C,\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{Q}:E}}{\mathtt{l}:C\multimap C,\mathtt{x}_r:C,\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{L}:E}}{\mathtt{l}:C\multimap C,\mathtt{r}:C\multimap C,\mathtt{q}:\mathbf{B}^n\vdash_{\mathrm{T}}\mathtt{let\ lN\ be\ t}_l,\mathtt{c}_l,\mathtt{b}_0^l\ \mathtt{in\ M}:E}}{\mathtt{z}:D\vdash_{\mathrm{T}}\mathtt{let\ z\ be\ l,r,q\ in\ M}:E}$$

and so we can conclude:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi\rhd\mathtt{c}:B\vdash_{\mathrm{T}}\mathtt{F}[\mathtt{c}]:\mathbf{B}\multimap C\multimap C}}{\mathtt{c}:!^iB\vdash_{\mathrm{T}}\mathtt{F}[\mathtt{c}]:!^i(\mathbf{B}\multimap C\multimap C)}\qquad\dfrac{}{\Sigma_1\rhd\mathtt{z}:D\vdash_{\mathrm{T}}\mathtt{let\ z\ be\ l,r,q\ in\ M}:E}}{\mathtt{s}:!^i(\mathbf{B}\multimap C\multimap C)\multimap D,\mathtt{c}:!^iB\vdash_{\mathrm{T}}\mathtt{let\ sF}[\mathtt{c}]\ \mathtt{be\ l,r,q\ in\ M}:E}}{\mathtt{s}:\mathbf{TM}_i,\mathtt{c}:!^iB\vdash_{\mathrm{T}}\mathtt{let\ s}(\mathtt{F}[\mathtt{c}])\ \mathtt{be\ l,r,q\ in\ M}:E}}{\mathtt{s}:\mathbf{TM}_i\vdash_{\mathrm{T}}\lambda\mathtt{c}.\mathtt{let\ s}(\mathtt{F}[\mathtt{c}])\ \mathtt{be\ l,r,q\ in\ M}:!^iB\multimap E}}{\mathtt{s}:\mathbf{TM}_i\vdash_{\mathrm{T}}\lambda\mathtt{c}.\mathtt{let\ s}(\mathtt{F}[\mathtt{c}])\ \mathtt{be\ l,r,q\ in\ M}:\mathbf{ID}_i}}{\vdash_{\mathrm{T}}\mathbf{Dec}:\mathbf{TM}_i\multimap\mathbf{ID}_i}$$

To verify that **Dec** has the intended behaviour is boring but easy. □

Analogously for the combining phase we have the following lemma.

Lemma 26. *The term:*

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

where $\mathtt{R} \doteq \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s.let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{c_r b'} \circ \mathtt{c_l b_l} \circ \mathtt{l, r, q'}\rangle$ *and* $\mathtt{L} \doteq \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s.let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{l, c_l b_l} \circ \mathtt{c_r b'} \circ \mathtt{r, q'}\rangle$*, is typable as:*

$$\vdash_{\mathrm{T}} \mathbf{Com} : \mathbf{ID}_i \multimap \mathbf{TM}_i$$

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

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

Proof. In order to type **Com** it is convenient to have some type abbreviations:

$$\begin{array}{lcllcl} A & = & \alpha \multimap \alpha & D & = & B \otimes A \otimes C \otimes \mathbf{B} \otimes C\\ B & = & A \otimes A \otimes \mathbf{B}^q & E & = & A \otimes A \otimes C \otimes \mathbf{B} \otimes C \otimes \mathbf{B} \otimes \mathbf{B}^q\\ C & = & \mathbf{B} \multimap \alpha \multimap \alpha & F & = & \mathbf{B} \otimes \mathbf{B}^q \otimes \mathbf{B}\end{array}$$

and some term abbreviations:

$$\begin{array}{lcllcl} \mathtt{M} & = & \mathtt{if\ m\ then\ R\ else\ L} & \mathtt{P} & = & \langle \mathtt{l, r, c_l, b_l, c_r, b_r, q}\rangle\\ \mathtt{N} & = & \langle \mathtt{l, r, c_l, b_l, c_r}\rangle & \mathtt{S} & = & \underline{\delta}\langle \mathtt{b_r, q}\rangle\\ \mathtt{Q} & = & \mathtt{c_r b'} \circ \mathtt{c_l b_l} \circ \mathtt{l, r} & & & \end{array}$$

Now let Σ be the following derivation:

$$\dfrac{\dfrac{\mathtt{b'} : \mathbf{B} \vdash_{\mathrm{T}} \mathtt{b'} : \mathbf{B} \quad \mathtt{g} : A \vdash_{\mathrm{T}} \mathtt{g} : A}{\mathtt{c_r} : C, \mathtt{b'} : \mathbf{B} \vdash_{\mathrm{T}} \mathtt{c_r b'} : A} \quad \dfrac{\mathtt{b_l} : \mathbf{B} \vdash_{\mathrm{T}} \mathtt{b_l} : \mathbf{B} \quad \mathtt{g} : A \vdash_{\mathrm{T}} \mathtt{g} : A}{\mathtt{c_l} : C, \mathtt{b_l} : \mathbf{B} \vdash_{\mathrm{T}} \mathtt{c_l b_l} : A} \quad \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{l} : A}{\mathtt{c_r} : C, \mathtt{c_l} : C, \mathtt{b_l} : \mathbf{B}, \mathtt{b'} : \mathbf{B}, \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{L} : A}$$

then we have the following derivation Π_R typing R

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma \rhd \mathtt{c_r} : C, \mathtt{c_l} : C, \mathtt{b_l} : \mathbf{B}, \mathtt{b'} : \mathbf{B}, \mathtt{l} : A \vdash_{\mathrm{T}} \mathtt{Q} : A \quad \mathtt{r} : A \vdash_{\mathrm{T}} \mathtt{r} : A}{\mathtt{c_r} : C, \mathtt{c_l} : C, \mathtt{b_l} : \mathbf{B}, \mathtt{b'} : \mathbf{B}, \mathtt{l} : A \mathtt{r} : A \vdash_{\mathrm{T}} \langle \mathtt{Q, r}\rangle : A \otimes A} \quad \mathtt{q'} : \mathbf{B}^q \vdash_{\mathrm{T}} \mathtt{q'} : \mathbf{B}^q}{\mathtt{c_r} : C, \mathtt{c_l} : C, \mathtt{b_l} : \mathbf{B}, \mathtt{b'} : \mathbf{B}, \mathtt{l} : A, \mathtt{r} : A, \mathtt{q'} : \mathbf{B}^q \vdash_{\mathrm{T}} \langle \mathtt{Q, r, q'}\rangle : B}}{\mathtt{s} : D, \mathtt{b'} : \mathbf{B}, \mathtt{q'} : \mathbf{B}^q \vdash_{\mathrm{T}} \mathtt{let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{Q, r, q'}\rangle : B}}{\mathtt{b'} : \mathbf{B}, \mathtt{q'} : \mathbf{B}^q \vdash_{\mathrm{T}} \lambda \mathtt{s.let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{Q, r, q'}\rangle : D \multimap B}}{\mathtt{b'} : \mathbf{B} \vdash_{\mathrm{T}} \lambda \mathtt{q'}.\lambda \mathtt{s.let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{Q, r, q'}\rangle : \mathbf{B}^q \multimap D \multimap B}}{\vdash_{\mathrm{T}} \lambda \mathtt{b'}.\lambda \mathtt{q'}.\lambda \mathtt{s.let\ s\ be\ l, r, c_l, b_l, c_r\ in}\ \langle \mathtt{Q, r, q'}\rangle : \mathbf{B} \multimap \mathbf{B}^q \multimap D \multimap B}$$

analogously we have a derivation Π_L with conclusion

$$\vdash_{\mathrm{T}} \mathtt{L} : \mathbf{B} \multimap \mathbf{B}^q \multimap D \multimap B$$

Now let Π be the following derivation:

$$\frac{\Pi_R \rhd \vdash_{\mathrm{T}} \mathtt{R} : \mathbf{B} \multimap \mathbf{B}^q \multimap D \multimap B \quad \Pi_L \rhd \vdash_{\mathrm{T}} \mathtt{L} : \mathbf{B} \multimap \mathbf{B}^q \multimap D \multimap B}{\mathtt{m} : \mathbf{B} \vdash_{\mathrm{T}} \mathtt{if\ m\ then\ R\ else\ L} : \mathbf{B} \multimap \mathbf{B}^q \multimap D \multimap B}$$

Furthermore, let $\Gamma = \{\mathtt{l} : A, \mathtt{r} : A, \mathtt{c}_l : C, \mathtt{b}_l : \mathbf{B}, \mathtt{c}_r : C\}$, then there is a derivation Σ_N with conclusion: $\Gamma \vdash_{\mathrm{T}} \mathtt{N} : D$. Hence it is easy to construct a derivation $\Theta \rhd \mathtt{m} : \mathbf{B}, \mathtt{b}' : \mathbf{B}, \mathtt{q}' : \mathbf{B}^q, \Gamma \vdash_{\mathrm{T}} \mathtt{Mb'q'N} : B$. By Lemma 21 there is a derivation Σ_R with conclusion: $\mathtt{b}_r : \mathbf{B}, \mathtt{q} : \mathbf{B}^q \vdash_{\mathrm{T}} \mathtt{S} : F$. Hence we can conclude:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{r} : C \vdash_{\mathrm{T}} \mathtt{r} : C}{\mathtt{r} :!^i C \vdash_{\mathrm{T}} \mathtt{r} :!^i C} \quad \dfrac{\Sigma_R \rhd \mathtt{b}_r : \mathbf{B}, \mathtt{q} : \mathbf{B}^q \vdash_{\mathrm{T}} \mathtt{S} : F \quad \dfrac{\Theta \rhd \mathtt{m} : \mathbf{B}, \mathtt{b}' : \mathbf{B}, \mathtt{q}' : \mathbf{B}^q, \Gamma \vdash_{\mathrm{T}} \mathtt{Mb'q'N} : B}{\mathtt{z} : F, \Gamma \vdash_{\mathrm{T}} \mathtt{let\ z\ be\ b', q', m\ in\ Mb'q'N} : B}}{\mathtt{b}_r : \mathbf{B}, \mathtt{q} : \mathbf{B}^q, \Gamma \vdash_{\mathrm{T}} \mathtt{let\ S\ be\ b', q', m\ in\ Mb'q'N} : B}}{\mathtt{z} : E \vdash_{\mathrm{T}} \mathtt{let\ z\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} : B}}{\mathtt{s} :!^i C \multimap E, \mathtt{c} :!^i C \vdash_{\mathrm{T}} \mathtt{let\ sc\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} : B}}{\mathtt{s} : \mathbf{ID}_i, \mathtt{c} :!^i C \vdash_{\mathrm{T}} \mathtt{let\ sc\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} : B}}{\mathtt{s} : \mathbf{ID}_i \vdash_{\mathrm{T}} \lambda \mathtt{c.let\ sc\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} :!^i C \multimap B}}{\mathtt{s} : \mathbf{ID}_i \vdash_{\mathrm{T}} \lambda \mathtt{c.let\ sc\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} : \mathbf{TM}_i}}{\vdash_{\mathrm{T}} \lambda \mathtt{s}.\lambda \mathtt{c.let\ sc\ be\ P\ in\ let\ S\ be\ b', q', m\ in\ Mb'q'N} : \mathbf{ID}_i \multimap \mathbf{TM}_i}$$

Again, checking that **Com** has the intended behaviour is boring but easy. □

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

Lemma 27. *The term* $\mathbf{Tr} \doteq \mathbf{Com} \circ \mathbf{Dec}$ *defines a Turing machine transition step and is typable as:*

$$\vdash_{\mathrm{T}} \mathbf{Tr} : \mathbf{TM}_i \multimap \mathbf{TM}_i$$

Proof. Easy, by Lemma 25 and Lemma 26. □

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

Lemma 28. *The term*

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

where $\mathtt{T} \doteq \lambda \mathtt{b}.\lambda \mathtt{s}.\lambda \mathtt{c.let\ sc\ be\ l, r, c}_l, \mathtt{b}_l, \mathtt{c}_r, \mathtt{b}_r, \mathtt{q\ in\ Rbq}\langle \mathtt{l, r, c}_l, \mathtt{b}_l, \mathtt{c}_r \rangle$ *and* $\mathtt{R} \doteq \lambda \mathtt{b}'.\lambda \mathtt{q}'.\lambda \mathtt{s.let\ s\ be\ l, r, c}_l, \mathtt{b}_l, \mathtt{c}_r\ \mathtt{in}\ \langle \mathtt{c}_r \mathtt{b}' \circ \mathtt{c}_l \mathtt{b}_l \circ \mathtt{l, r, q}' \rangle$*, defines the function that, when supplied by a boolean string and a Turing machine, writes the input string on the tape of the Turing machine. Such a term is typable as*

$$\vdash_{\mathrm{T}} \mathbf{In} : \mathbf{S} \multimap \mathbf{TM}_i \multimap \mathbf{TM}_i$$

Proof. Similar to the proof of Lemma 26. Note that for the term $\mathtt{T}$ we have the typing: $\vdash_{\mathrm{T}} \mathtt{T} : \mathbf{B} \multimap \mathbf{ID}_i \multimap \mathbf{TM}_i$. □

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

Lemma 29. *Let f be a function deciding if a state is accepting or rejecting. Then the term:*

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

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

$$\vdash_{\mathrm{T}} \mathbf{Ext} : \mathbf{TM}_i \multimap \mathbf{B}$$

Proof. Easy. Note that the existence of the term $\underline{f}$ is assured by Lemma 21. □

Now we can show that STA is complete for PTIME.

Theorem 8 (PTIME Completeness). *Let a decision problem $\mathcal{P}$ be decided in* polynomial time *P, where $deg(P) = m$, and in* polynomial space *Q, where $deg(Q) = l$, by a Turing machine $\mathcal{M}$. Then it is definable by a term $\underline{\mathtt{M}}$ typable in* STA *as:*

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

Proof. By Lemma 18: $\mathtt{s}_p :!^m\mathbf{S} \vdash_{\mathrm{T}} \underline{\mathtt{P}}[\mathbf{lens}_p/\mathtt{x}] : \mathbf{N}_{2m+1}$ and $\mathtt{s}_q :!^l\mathbf{S} \vdash_{\mathrm{T}} \underline{\mathtt{Q}}[\mathbf{lens}_q/x] : \mathbf{N}_{2l+1}$. Furthermore, by composition:

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

so by (cut) and by some applications of (m) rule the conclusion follows. □

Without loss of generality we can assume that a Turing machine stops on accepting states with the head at the begin of the tape. The following lemma shows that the function extracting the output string from the final configuration is easily definable.

Lemma 30. *The term:*

$$\mathbf{Ext_F} \doteq \lambda \mathtt{sc.let\ sc\ be\ l, r, q\ in\ r}$$

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

$$\vdash_{\mathrm{T}} \mathbf{Ext_F} : \mathbf{TM}_i \multimap \mathbf{S}_i$$

Proof. Easy. □

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

Theorem 9 (FPTIME Completeness). *Let a function $\mathcal{F}$ be computed in* polynomial time *P, where $deg(P) = m$, and in* polynomial space *Q, where $deg(Q) = l$, by a Turing machine $\mathcal{M}$. Then it is definable by a term $\underline{\mathtt{M}}$ typable in* STA *as:*

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

Proof. Similar to the proof of Theorem 8 but using $\mathbf{Ext_F}$. By Lemma 18: $\mathtt{s}_p :!^m\mathbf{S} \vdash_{\mathtt{T}} \underline{\mathtt{P}}[\mathbf{lens}_p/\mathtt{x}] : \mathbf{N}_{2m+1}$ and $\mathtt{s}_q :!^l\mathbf{S} \vdash_{\mathtt{T}} \underline{\mathtt{Q}}[\mathbf{lens}_q/x] : \mathbf{N}_{2l+1}$. Furthermore by composition:

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

so by (cut) and some applications of (m) rule the conclusion follows. □

3. Soft Type Assignment systems in Natural Deduction style

3.1. Introduction

In this chapter we present two Natural Deduction versions for the Soft Type Assignment system presented in Chapter 2.
Firstly, we introduce the system $\mathrm{STA}_{\mathrm{N}}$ and we consider the equivalence between $\mathrm{STA}_{\mathrm{N}}$ and STA. We prove that the two systems are equivalent with respect to typability power and also with respect to complexity properties. In fact, we show that for every $\mathrm{STA}_{\mathrm{N}}$ derivation there is a STA derivation, with the same conclusion and measures, and vice-versa.
Then we introduce the system $\mathrm{STA}_{\mathrm{M}}$, a natural deduction systems where contexts are multisets. $\mathrm{STA}_{\mathrm{M}}$ differs from $\mathrm{STA}_{\mathrm{N}}$ since multisets allow the reuse of variable names and avoid the use of rules renaming variables in the subject. We prove the equivalence between $\mathrm{STA}_{\mathrm{M}}$ and $\mathrm{STA}_{\mathrm{N}}$. Derivations in the two systems have the same structure, and so the same complexity properties, but the proof involves some technicalities. To overcome to some of them we introduce as a technical tool a further system where contexts are lists.

3.2. A Natural Deduction version of the Soft Type Assignment system

In this section we present the system $\mathrm{STA}_{\mathrm{N}}$. To make the treatment complete, we consider the equivalence between $\mathrm{STA}_{\mathrm{N}}$ and STA and we prove it in two steps.
Firstly, in a standard way, we prove that the two systems are equivalent, with respect to typability power. This equivalence is not sufficient. In fact the complexity bound expressed by Theorem 7, which we would adapt to terms typable in $\mathrm{STA}_{\mathrm{N}}$, depends, not only on types and terms, but also on the type derivation structure.
For this reason, as a second step, we show that the measures defined over STA derivation

Axiom and Weakening:

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

Multiplicatives:

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

Exponentials:

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

Quantifier:

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

Table 3.1.: Natural Deduction version of STA.

in Section 2.4 can be straightforward defined over $\mathrm{STA}_{\mathrm{N}}$ derivations. Hence we extend the proof of the equivalence between the two systems by considering measures over derivations.
In Chapter 5 $\mathrm{STA}_{\mathrm{N}}$ will be used as the core for a language characterizing polynomial space computations.

3.2.1. The system $\mathrm{STA}_{\mathrm{N}}$

$\mathrm{STA}_{\mathrm{N}}$ proves sequents of the shape:

$$\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$$

where Γ is a context, $\mathtt{M}$ is a λ-term and $\sigma \in \mathcal{T}$ is a soft type as defined in Definition 4. The rules defining $\mathrm{STA}_{\mathrm{N}}$ are depicted in Table 3.1. $\mathrm{STA}_{\mathrm{N}}$ contexts are defined analogously to STA contexts. For $\mathrm{STA}_{\mathrm{N}}$ contexts and derivations we will use the same notation used in Chapter 2 for STA contexts and derivations respectively. We hope that it will be clear from the context which system we refer to.
$\mathrm{STA}_{\mathrm{N}}$ enjoys the following standard properties for a natural deduction system.

Lemma 31.

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

Proof.
All the three point can be proved by induction on the derivation proving $\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$. Base cases are trivial, the others follow directly by induction hypothesis. □

In order to prove in an easy way the equivalence between STA and $\mathrm{STA}_{\mathrm{N}}$ it will be useful the following weaker form of the Substitution Lemma.

Lemma 32 (Linear Substitution Lemma).
Let $\Gamma, \mathtt{x} : A \vdash_{\mathrm{N}} \mathtt{M} : \sigma$ *and* $\Delta \vdash_{\mathrm{N}} \mathtt{N} : A$ *where* $\Gamma \# \Delta$. *Then*

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

Proof. By induction on the derivation Π proving $\Gamma, \mathtt{x} : A \vdash_{\mathrm{N}} \mathtt{M} : \sigma$. The base case is trivial. The case where Π ends by (sp) rule is not possible. The cases where Π ends

either by rule $(\multimap I), (m), (\forall I)$ or $(\forall E)$ follow directly by induction hypothesis. The case where Π ends as:

$$\frac{\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : A \vdash_{\mathrm{N}} \mathtt{M} : \sigma}\ (w)$$

follows trivially by Lemma 31.2. Let now Π ends as:

$$\frac{\Gamma_1 \vdash_{\mathrm{N}} \mathtt{M}' : \tau \multimap B \quad \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}' : \tau \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'\mathtt{N}' : B}\ (\multimap E)$$

then by the side condition $\Gamma_1 \# \Gamma_2$ either, $\mathtt{x} : A \in \Gamma_1$ and $\mathtt{x} \notin \mathrm{FV}(\mathtt{N}')$, or $\mathtt{x} : A \in \Gamma_2$ and $\mathtt{x} \notin \mathrm{FV}(\mathtt{M}')$. In the first case by induction hypothesis we have $\Delta, \Gamma_1 \vdash_{\mathrm{N}} \mathtt{M}'[\mathtt{N}/\mathtt{x}] : \tau \multimap B$ so by applying a $(\multimap E)$ rule we conclude: $\Delta, \Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'[\mathtt{N}/\mathtt{x}]\mathtt{N}' \equiv (\mathtt{M}'\mathtt{N}')[\mathtt{N}/\mathtt{x}] : B$. In the second case by induction hypothesis we have $\Delta, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}'[\mathtt{N}/\mathtt{x}] : \tau$ and by applying a $(\multimap E)$ rule we conclude: $\Delta, \Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'\mathtt{N}'[\mathtt{N}/\mathtt{x}] \equiv (\mathtt{M}'\mathtt{N}')[\mathtt{N}/\mathtt{x}] : B$. □

3.2.2. Equivalence between $\mathrm{STA}_{\mathrm{N}}$ and STA

We would now prove the equivalence, with respect to the typability power, between $\mathrm{STA}_{\mathrm{N}}$ and STA. The following lemma shows that $\mathrm{STA}_{\mathrm{N}}$ has at least the same typability power as STA.

Lemma 33.

$$\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma \textit{ implies } \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$$

Proof. By induction on the derivation Π proving $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. Base case is trivial. The cases where Π ends either by rule $(w), (\multimap R), (m), (sp)$ or $(\forall R)$ follow directly by induction hypothesis. The case Π ends by the (cut) rule follows directly by Lemma 32. Consider the case Π ends by the rule:

$$\frac{\Gamma \vdash_{\mathrm{T}} \mathtt{N} : \tau \quad \mathtt{x} : A, \Delta \vdash_{\mathrm{T}} \mathtt{P} : \sigma}{\Gamma, \mathtt{y} : \tau \multimap A, \Delta \vdash_{\mathrm{T}} \mathtt{P}[\mathtt{y}\mathtt{N}/\mathtt{x}] : \sigma}\ (\multimap L)$$

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

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

and since by induction hypothesis we also have $\mathtt{x} : A, \Delta \vdash_{\mathrm{N}} \mathtt{P} : \sigma$, by applying Lemma 32 the conclusion follows.
Consider the case Π ends by the rule:

$$\frac{\Gamma, \mathtt{x} : A[B/\alpha] \vdash_{\mathrm{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : \forall\alpha.A \vdash_{\mathrm{T}} \mathtt{M} : \sigma}\ (\forall L)$$

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

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

by applying Lemma 32 the conclusion follows. □

Moreover, the following lemma shows that STA has at least the same typability power as $\mathrm{STA}_{\mathrm{N}}$.

Lemma 34.

$$\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma \ \textit{implies}\ \Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$$

Proof. By induction on the derivation Π proving $\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$. Base case is trivial. The cases where Π ends either by rule $(w), (\multimap I), (m), (sp)$ or $(\forall I)$ follow directly by induction hypothesis.
Consider the case Π ends by the rule:

$$\dfrac{\Delta \vdash_{\mathrm{N}} \mathtt{P} : \tau \multimap A \quad \Gamma \vdash_{\mathrm{N}} \mathtt{N} : \tau}{\Gamma, \Delta \vdash_{\mathrm{N}} \mathtt{PN} : A}\,(\multimap E)$$

By induction hypothesis we have derivations with conclusion $\Delta \vdash_{\mathrm{T}} \mathtt{P} : \tau \multimap A$ and $\Gamma \vdash_{\mathrm{T}} \mathtt{N} : \tau$ hence we can conclude by constructing a derivation ending as:

$$\dfrac{\Delta \vdash_{\mathrm{T}} \mathtt{P} : \tau \multimap A \quad \dfrac{\Gamma \vdash_{\mathrm{T}} \mathtt{N} : \tau \quad \dfrac{}{\mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{x} : A}\,(Ax)}{\Gamma, \mathtt{y} : \tau \multimap A \vdash_{\mathrm{T}} \mathtt{x}[\mathtt{yN}/\mathtt{x}] : \sigma}\,(\multimap L)}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{PN} : A}\,(cut)$$

Consider the case Π ends by the rule:

$$\dfrac{\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \forall\alpha.A}{\Gamma \vdash_{\mathrm{N}} \mathtt{M} : A[B/\alpha]}\,(\forall E)$$

By induction hypothesis we have a derivation with conclusion $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \forall\alpha.A$ hence we can conclude by constructing a derivation ending as:

$$\dfrac{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \forall\alpha.A \quad \dfrac{\mathtt{x} : A[B/\alpha] \vdash_{\mathrm{T}} \mathtt{x} : A[B/\alpha]}{\mathtt{x} : \forall\alpha.A \vdash_{\mathrm{T}} \mathtt{x} : A[B/\alpha]}\,(\forall L)}{\Gamma \vdash_{\mathrm{T}} \mathtt{M} : A[B/\alpha]}\,(cut)$$

□

Then, we can prove the following theorem.

Theorem 10.

$$\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma \iff \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$$

Proof. By Lemma 33 and Lemma 34. □

By the above theorem it follows that also for $\mathrm{STA}_{\mathrm{N}}$ they hold the substitution and the subject reduction properties.

Lemma 35 (Substitution Lemma). *Let* $\Gamma, \mathtt{x} : \tau \vdash_{\mathrm{N}} \mathtt{M} : \sigma$ *and* $\Delta \vdash_{\mathrm{N}} \mathtt{N} : \tau$ *where* $\Gamma \# \Delta$. *Then:*

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

Proof. By Theorem 10 and Lemma 10. □

Theorem 11 (Subject Reduction). *If* $\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_\beta \mathtt{M}'$ *then* $\Gamma \vdash_{\mathrm{N}} \mathtt{M}' : \sigma$

Proof. By Theorem 10 and Theorem 5. □

3.2.3. Measure Equivalence

The relation between STA and $\mathrm{STA}_{\mathrm{N}}$ goes beyond the equivalence stated by Theorem 10. Here, we show that the measures defined over STA derivations in Section 2.4 can be equivalently defined over $\mathrm{STA}_{\mathrm{N}}$ derivations. So, the equivalence between STA and $\mathrm{STA}_{\mathrm{N}}$ can be reformulated considering also this measures. In fact, the results that follow are the analogous of the results of the previous section where measures are considered.

Definition 12.

- *The* rank *of a rule* (m) *as:*

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

  is the number $k \leq n$ *of variables* $\mathtt{x}_i$ *such that* $\mathtt{x}_i \in \mathrm{FV}(\mathtt{M})$ *for* $1 \leq i \leq n$.
  Let r *be the the maximum rank of a rule* (m) *in* Π. *The* rank $\mathtt{rk}(\Pi)$ *of* Π *is the maximum between* 1 *and* r.

- *The* degree $\mathtt{d}(\Pi)$ *of* Π *is the maximum nesting of applications of rule* (sp) *in* Π.

- *Let* r *be a natural number. The* weight $\mathtt{W}(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows.*

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

- *If the last applied rule is:*

$$\frac{\Sigma \rhd \Gamma, \mathtt{x} : \sigma \vdash_{\mathrm{N}} \mathtt{M} : A}{\Gamma \vdash_{\mathrm{N}} \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A} \ (\multimap R)$$

  then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + 1$.

- *If the last applied rule is:*

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

  then $\mathtt{W}(\Pi, r) = r\mathtt{W}(\Sigma, r)$.

- *If the last applied rule is:*

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

  then $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) + \delta(\Theta, r)$.

- *In every other case* $\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r)$ *where* Σ *is the unique premise derivation.*

Now, we are ready to prove that for every $\mathrm{STA}_{\mathrm{N}}$ derivation there exists an STA derivation, with the same conclusion and measures, and vice-versa.

Lemma 36. *Let* Π *be a derivation in* $\mathrm{STA}_{\mathrm{N}}$ *proving* $\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$. *Then there exists a derivation* Σ *in* STA *proving* $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ *such that for every* $r \in \mathbb{N}$*:*

$$\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) \qquad \mathtt{rk}(\Pi) = \mathtt{rk}(\Sigma) \qquad \mathtt{d}(\Pi) = \mathtt{d}(\Sigma)$$

Proof. The proof is analogous to the proof of Lemma 34 but considering how the measures are modified.
By induction on Π. Base case is trivial. The cases where Π end either by rule $(\multimap I), (sp), (m), (\forall I)$ or (w) follows directly by induction hypothesis and weight definition. Consider the case Π ends as:

$$\frac{\Pi_1 \rhd \Delta \vdash_{\mathrm{N}} \mathtt{P} : \tau \multimap A \quad \Pi_2 \rhd \Gamma \vdash_{\mathrm{N}} \mathtt{N} : \tau}{\Gamma, \Delta \vdash_{\mathrm{N}} \mathtt{PN} : A} \ (\multimap E)$$

By induction hypothesis we have derivations $\Sigma_1 \rhd \Delta \vdash_{\mathrm{T}} \mathtt{P} : \tau \multimap A$ and $\Sigma_2 \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{N} : \tau$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Pi_1, r) = \mathtt{W}(\Sigma_1, r)$, $\mathtt{W}(\Pi_2, r) = \mathtt{W}(\Sigma_2, r)$, $\mathtt{rk}(\Pi_1) = \mathtt{rk}(\Sigma_1)$,$\mathtt{rk}(\Pi_2) = \mathtt{rk}(\Sigma_2)$, $\mathtt{d}(\Pi_1) = \mathtt{d}(\Sigma_1)$ and $\mathtt{d}(\Pi_2) = \mathtt{d}(\Sigma_2)$. Hence we can build a derivation Σ ending as:

$$\dfrac{\Sigma_1 \rhd \Delta \vdash_{\mathrm{T}} \mathtt{P} : \tau \multimap A \qquad \dfrac{\Sigma_2 \rhd \Gamma \vdash_{\mathrm{T}} \mathtt{N} : \tau \quad \dfrac{}{\mathtt{x} : A \vdash_{\mathrm{T}} \mathtt{x} : A}\,(Ax)}{\Gamma, \mathtt{y} : \tau \multimap A \vdash_{\mathrm{T}} \mathtt{x}[\mathtt{yN}/\mathtt{x}] : \sigma}\,(\multimap L)}{\Gamma, \Delta \vdash_{\mathrm{T}} \mathtt{PN} : A}\,(cut)$$

and by measures definition:

$$\mathtt{W}(\Sigma, r) = (\mathtt{W}(\Sigma_1, r)+(\mathtt{W}(\Sigma_2, r)+1))-1 = \mathtt{W}(\Sigma_1, r)+\mathtt{W}(\Sigma_2, r) = \mathtt{W}(\Pi_1, r)+\mathtt{W}(\Pi_2, r) = \mathtt{W}(\Pi, r)$$

$$\mathtt{rk}(\Sigma) = \max(\mathtt{rk}(\Sigma_1), \mathtt{rk}(\Sigma_2)) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Pi_2)) = \mathtt{rk}(\Pi)$$

$$\mathtt{d}(\Sigma) = \max(\mathtt{d}(\Sigma_1), \mathtt{d}(\Sigma_2)) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Pi_2)) = \mathtt{d}(\Pi)$$

Analogously consider the case Π ends by the rule:

$$\frac{\Pi_1 \rhd \Gamma \vdash_{\mathtt{N}} \mathtt{M} : \forall\alpha.A}{\Gamma \vdash_{\mathtt{N}} \mathtt{M} : A[B/\alpha]}\ (\forall E)$$

By induction hypothesis we have a derivation $\Sigma_1 \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M} : \forall\alpha.A$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Pi_1, r) = \mathtt{W}(\Sigma_1, r)$, $\mathtt{rk}(\Pi_1) = \mathtt{rk}(\Sigma_1)$ and $\mathtt{d}(\Pi_1) = \mathtt{d}(\Sigma_1)$. Hence we can build a derivation Σ ending as:

$$\frac{\Sigma_1 \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{M} : \forall\alpha.A \qquad \dfrac{\mathtt{x} : A[B/\alpha] \vdash_{\mathtt{T}} \mathtt{x} : A[B/\alpha]}{\mathtt{x} : \forall\alpha.A \vdash_{\mathtt{T}} \mathtt{x} : A[B/\alpha]}\ (\forall L)}{\Gamma \vdash_{\mathtt{T}} \mathtt{M} : A[B/\alpha]}\ (cut)$$

and by measures definition:

$$\mathtt{W}(\Sigma, r) = (\mathtt{W}(\Sigma_1, r) + 1) - 1 = \mathtt{W}(\Sigma_1, r) = \mathtt{W}(\Pi_1, r) = \mathtt{W}(\Pi)$$

$$\mathtt{rk}(\Sigma) = \mathtt{rk}(\Sigma_1) = \mathtt{rk}(\Pi_1) = \mathtt{rk}(\Pi)$$

$$\mathtt{d}(\Sigma) = \mathtt{d}(\Sigma_1) = \mathtt{d}(\Pi_1) = \mathtt{d}(\Pi)$$

□

In order to prove the converse of the above lemma we need the following version of the Linear Substitution Lemma, where measures are considered.

Lemma 37 (Weighted Linear Substitution Lemma).
Let $\Pi \rhd \Gamma, \mathtt{x} : A \vdash_{\mathtt{N}} \mathtt{M} : \sigma$ *and* $\Sigma \rhd \Delta \vdash_{\mathtt{N}} \mathtt{N} : A$ *where* $\Gamma \# \Delta$. *Then there exists a derivation* Θ *proving:*

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

such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Theta, r) = \mathtt{W}(\Pi, r)+\mathtt{W}(\Sigma, r)-1$, $\mathtt{rk}(\Theta) = \max(\mathtt{rk}(\Pi), \mathtt{rk}(\Sigma))$ *and* $\mathtt{d}(\Theta) = \max(\mathtt{d}(\Pi), \mathtt{d}(\Sigma))$.

Proof. The proof is analogous to the proof of Lemma 32, but considering how the measures are modified.
By induction on Π. The base case is trivial. The case where Π ends by (sp) rule is not possible. The cases where Π ends either by rule $(\multimap I), (m), (w), (\forall I)$ or $(\forall E)$ follow directly by induction hypothesis and measures definition. Let now Π ends as:

$$\frac{\Pi_1 \rhd \Gamma_1 \vdash_{\mathrm{N}} \mathtt{M}' : \tau \multimap B \quad \Pi_2 \rhd \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}' : \tau \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'\mathtt{N}' : B} \ (\multimap E)$$

then by the side condition $\Gamma\#\Delta$ either, $\mathtt{x} : A \in \Gamma_1$ and $\mathtt{x} \notin \mathrm{FV}(\mathtt{N}')$, or $\mathtt{x} : A \in \Gamma_2$ and $\mathtt{x} \notin \mathrm{FV}(\mathtt{M}')$. In the first case by induction hypothesis we have a derivation Θ_1 proving $\Delta, \Gamma_1 \vdash_{\mathrm{N}} \mathtt{M}'[\mathtt{N}/\mathtt{x}] : \tau \multimap B$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Theta_1, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Sigma, r) - 1$, $\mathtt{rk}(\Theta_1) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Sigma))$ and $\mathtt{d}(\Theta_1) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Sigma))$. So we can build Σ as:

$$\frac{\Theta_1 \rhd \Delta, \Gamma_1 \vdash_{\mathrm{N}} \mathtt{M}'[\mathtt{N}/\mathtt{x}] : \tau \multimap B \quad \Pi_2 \rhd \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}' : \tau}{\Delta, \Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'[\mathtt{N}/\mathtt{x}]\mathtt{N}' \equiv (\mathtt{M}'\mathtt{N}')[\mathtt{N}/\mathtt{x}] : B} \ (\multimap E)$$

and by measures definition:

$$\mathtt{W}(\Theta, r) = \mathtt{W}(\Theta_1, r) + \mathtt{W}(\Pi_2, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Sigma, r) - 1 = \mathtt{W}(\Pi, r) + \mathtt{W}(\Sigma, r) - 1$$

$$\mathtt{rk}(\Theta) = \max(\mathtt{rk}(\Theta_1), \mathtt{rk}(\Pi_2)) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Sigma), \mathtt{rk}(\Pi_2)) = \max(\mathtt{rk}(\Pi), \mathtt{rk}(\Sigma))$$

$$\mathtt{d}(\Theta) = \max(\mathtt{d}(\Theta_1), \mathtt{d}(\Pi_2)) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Sigma), \mathtt{d}(\Pi_2)) = \max(\mathtt{d}(\Pi), \mathtt{d}(\Sigma))$$

In the second case by induction hypothesis we have a derivation Θ_1 proving $\Delta, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}'[\mathtt{N}/\mathtt{x}] : \tau$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Theta_1, r) = \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Sigma, r) - 1$, $\mathtt{rk}(\Theta_1) = \max(\mathtt{rk}(\Sigma), \mathtt{rk}(\Pi_2))$ and $\mathtt{d}(\Theta_1) = \max(\mathtt{d}(\Sigma), \mathtt{d}(\Pi_2))$. so we can build Σ as:

$$\frac{\Pi_1 \rhd \Gamma_1 \vdash_{\mathrm{N}} \mathtt{M} : \tau \quad \Theta_1 \rhd \Delta, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{N}'[\mathtt{N}/\mathtt{x}] : \tau}{\Delta, \Gamma_1, \Gamma_2 \vdash_{\mathrm{N}} \mathtt{M}'\mathtt{N}'[\mathtt{N}/\mathtt{x}] \equiv (\mathtt{M}'\mathtt{N}')[\mathtt{N}/\mathtt{x}] : B} \ (\multimap E)$$

and by measures definition:

$$\mathtt{W}(\Theta, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Theta_1, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) + \mathtt{W}(\Sigma, r) - 1 = \mathtt{W}(\Pi, r) + \mathtt{W}(\Sigma, r) - 1$$

$$\mathtt{rk}(\Theta) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Theta_1)) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Sigma), \mathtt{rk}(\Pi_2)) = \max(\mathtt{rk}(\Pi), \mathtt{rk}(\Sigma))$$

$$\mathtt{d}(\Theta) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Theta_1)) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Sigma), \mathtt{d}(\Pi_2)) = \max(\mathtt{d}(\Pi), \mathtt{d}(\Sigma))$$

□

Lemma 38. *Let Π be a derivation in* STA *proving $\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$. Then there exists a derivation Σ in $\mathrm{STA}_{\mathrm{N}}$ proving $\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$ such that for every $r \in \mathbb{N}$::*

$$\mathtt{W}(\Pi, r) = \mathtt{W}(\Sigma, r) \qquad \mathtt{rk}(\Pi) = \mathtt{rk}(\Sigma) \qquad \mathtt{d}(\Pi) = \mathtt{d}(\Sigma)$$

Proof. The proof is analogous to the proof of Lemma 33, but considering how the measures are modified.
By induction on Π. Base case is trivial. The cases Π end either by rule $(w), (\multimap R), (m), (sp)$ or $(\forall R)$ follow directly by induction hypothesis. The case Π ends by the (cut) rule follows directly by Lemma 37. Consider the case Π ends as:

$$\frac{\Pi_1 \rhd \Gamma \vdash_{\mathtt{T}} \mathtt{N} : \tau \quad \Pi_2 \rhd \mathtt{x} : A, \Delta \vdash_{\mathtt{T}} \mathtt{P} : \sigma}{\Gamma, \mathtt{y} : \tau \multimap A, \Delta \vdash_{\mathtt{T}} \mathtt{P}[\mathtt{yN}/\mathtt{x}] : \sigma} \ (\multimap L)$$

By induction hypothesis we have derivations $\Sigma_1 \rhd \Gamma \vdash_{\mathtt{N}} \mathtt{N} : \tau$ and $\Sigma_2 \rhd \mathtt{x} : A, \Delta \vdash_{\mathtt{N}} \mathtt{P} : \sigma$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Pi_1, r) = \mathtt{W}(\Sigma_1, r)$, $\mathtt{W}(\Pi_2, r) = \mathtt{W}(\Sigma_2, r)$, $\mathtt{rk}(\Pi_1) = \mathtt{rk}(\Sigma_1)$,$\mathtt{rk}(\Pi_2) = \mathtt{rk}(\Sigma_2)$, $\mathtt{d}(\Pi_1) = \mathtt{d}(\Sigma_1)$ and $\mathtt{d}(\Pi_2) = \mathtt{d}(\Sigma_2)$. Hence we can build a derivation Σ ending as:

$$\frac{\dfrac{}{\mathtt{y} : \tau \multimap A \vdash_{\mathtt{N}} \mathtt{y} : \tau \multimap A} \ (Ax) \quad \Sigma_1 \rhd \Gamma \vdash_{\mathtt{N}} \mathtt{N} : \tau}{\Gamma, \mathtt{y} : \tau \multimap A \vdash_{\mathtt{N}} \mathtt{yN} : A} \ (\multimap E)$$

such that $\mathtt{W}(\Theta_1, r) = \mathtt{W}(\Sigma_1, r) + 1$, $\mathtt{rk}(\Theta_1) = \mathtt{rk}(\Sigma_1)$ and $\mathtt{d}(\Theta_1) = \mathtt{d}(\Sigma_1)$. So by Lemma 37 we have $\Sigma \rhd \Gamma, \mathtt{y} : \tau \multimap A, \Delta \vdash_{\mathtt{N}} \mathtt{P}[\mathtt{yN}/\mathtt{x}] : \sigma$ such that:

$$\mathtt{W}(\Sigma, r) = \mathtt{W}(\Theta_1, r) + \mathtt{W}(\Sigma_2, r) - 1 = \mathtt{W}(\Sigma_1, r) + \mathtt{W}(\Sigma_2, r) = \mathtt{W}(\Pi_1, r) + \mathtt{W}(\Pi_2, r) = \mathtt{W}(\Pi, r)$$

$$\mathtt{rk}(\Sigma) = \max(\mathtt{rk}(\Theta_1), \mathtt{rk}(\Sigma_2)) = \max(\mathtt{rk}(\Sigma_1), \mathtt{rk}(\Sigma_2)) = \max(\mathtt{rk}(\Pi_1), \mathtt{rk}(\Pi_2)) = \mathtt{rk}(\Pi)$$

$$\mathtt{d}(\Sigma) = \max(\mathtt{d}(\Theta_1), \mathtt{d}(\Sigma_2)) = \max(\mathtt{d}(\Sigma_1), \mathtt{d}(\Sigma_2)) = \max(\mathtt{d}(\Pi_1), \mathtt{d}(\Pi_2)) = \mathtt{d}(\Pi)$$

Analogously consider the case Π ends as:

$$\frac{\Pi_1 \rhd \Gamma, \mathtt{x} : A[B/\alpha] \vdash_{\mathtt{T}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : \forall \alpha.A \vdash_{\mathtt{T}} \mathtt{M} : \sigma} \ (\forall L)$$

By induction hypothesis we have a derivation $\Sigma_1 \rhd \Gamma, \mathtt{x} : A[B/\alpha] \vdash_{\mathtt{N}} \mathtt{M} : \sigma$ such that for every $r \in \mathbb{N}$: $\mathtt{W}(\Pi_1, r) = \mathtt{W}(\Sigma_1, r)$, $\mathtt{rk}(\Pi_1) = \mathtt{rk}(\Sigma_1)$ and $\mathtt{d}(\Pi_1) = \mathtt{d}(\Sigma_1)$. Moreover we can build a derivation Θ_1 as:

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

such that $\mathtt{W}(\Theta_1, r) = 1$, $\mathtt{rk}(\Theta_1) = 1$ and $\mathtt{d}(\Theta_1) = 0$. So by Lemma 37 we have $\Sigma \rhd \Gamma, \mathtt{x} : \forall \alpha.A \vdash_{\mathtt{N}} \mathtt{M} : \sigma$ such that:

$$\mathtt{W}(\Sigma, r) = \mathtt{W}(\Theta_1, r) + \mathtt{W}(\Sigma_1, r) - 1 = \mathtt{W}(\Sigma_1, r) = \mathtt{W}(\Pi_1, r) = \mathtt{W}(\Pi, r)$$

$$\mathtt{rk}(\Sigma) = \max(\mathtt{rk}(\Theta_1), \mathtt{rk}(\Sigma_1)) = \mathtt{rk}(\Sigma_1) = \mathtt{rk}(\Pi_1) = \mathtt{rk}(\Pi)$$

$$\mathtt{d}(\Sigma) = \max(\mathtt{d}(\Theta_1), \mathtt{d}(\Sigma_1)) = \mathtt{d}(\Sigma_1) = \mathtt{d}(\Pi_1) = \mathtt{d}(\Pi)$$

□

From the above Lemmas it follows that the polytime soundness and the FPTIME completeness theorems holds also for $\mathrm{STA}_{\mathrm{N}}$.

Theorem 12 (Polytime Soundness of $\mathrm{STA}_{\mathrm{N}}$). *Let $\Pi \rhd \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$, then $\mathtt{M}$ can be evaluated to normal form on a Turing machine in time $O(|\mathtt{M}|^{3(\mathrm{d}(\Pi)+1)})$.*

Proof. By Lemma 36, Lemma 38 and Theorem 7. □

Theorem 13 (FPTIME Completeness of $\mathrm{STA}_{\mathrm{N}}$). *Let a function $\mathcal{F}$ be computed in* polynomial time *P, where $deg(P) = m$, and in* polynomial space *Q, where $deg(Q) = l$, by a Turing machine $\mathcal{M}$. Then it is definable by a term $\underline{\mathtt{M}}$ typable in $\mathrm{STA}_{\mathrm{N}}$ as:*

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

Proof. By Theorem 10 and Theorem 9. □

3.3. A Multiset Natural Deduction Soft Type Assignment system

In this section we introduce the system $\mathrm{STA}_{\mathrm{M}}$. It differs from the natural deduction system $\mathrm{STA}_{\mathrm{N}}$ presented in the previous section since it avoids the use of rules renaming variables in the subject, i.e. notably the (m) rule in Table 3.1.
The key idea underlying $\mathrm{STA}_{\mathrm{M}}$ is to use as contexts, instead of sets, multisets of type assignments. Multisets allow the reuse of variable names, hence renaming is no more necessary. So the (m) rule in $\mathrm{STA}_{\mathrm{M}}$ is used only for collapsing a multiset into a set.
$\mathrm{STA}_{\mathrm{M}}$ has the advantage that subjects are built just by three rules (axiom, introduction and elimination of the arrow) corresponding to the syntax formation rules of terms. Thanks to this property $\mathrm{STA}_{\mathrm{M}}$ seems a simpler framework to study the design of a type inference algorithm.
While clear in an intuitive way, the proof of the equivalence between $\mathrm{STA}_{\mathrm{M}}$ and $\mathrm{STA}_{\mathrm{N}}$ involves some technicalities. Derivations in the two systems have the same structure, and so the same complexity properties, but to pass to $\mathrm{STA}_{\mathrm{M}}$ from $\mathrm{STA}_{\mathrm{N}}$ is necessary to identify occurrences of variables. Analogously, to prove the converse is necessary to distinguish occurrences of variables. For this reason we introduce as a technical tool a further system where contexts are ordered multisets, i.e. lists.

3.3.1. The system $\mathrm{STA}_{\mathrm{M}}$

$\mathrm{STA}_{\mathrm{M}}$ proves sequents of the shape:

$$\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma$$

Axiom and Weakening:

$$\frac{}{\mathtt{x} : A \vdash_{\mathrm{M}} \mathtt{x} : A}\ (Ax) \qquad \frac{\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma}{\mathfrak{A}, \mathtt{x} : A \vdash_{\mathrm{M}} \mathtt{M} : \sigma}\ (w)$$

Multiplicatives:

$$\frac{\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma \multimap A \quad \mathfrak{B} \vdash_{\mathrm{M}} \mathtt{N} : \sigma \quad \mathfrak{A} \approx \mathfrak{B}}{\mathfrak{A}, \mathfrak{B} \vdash_{\mathrm{M}} \mathtt{MN} : A}\ (\multimap E) \qquad \frac{\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathrm{M}} \mathtt{M} : A \quad \mathtt{x} \notin \mathrm{dom}(\mathfrak{A})}{\mathfrak{A} \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap I)$$

Exponentials:

$$\frac{\mathfrak{A}, (\mathtt{x} : \tau)^{(r)} \vdash_{\mathrm{M}} \mathtt{M} : \sigma}{\mathfrak{A}, \mathtt{x} :! \tau, \vdash_{\mathrm{M}} \mathtt{M} : \sigma}\ (m) \qquad \frac{\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma}{!\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} :! \sigma}\ (sp)$$

Quantifier:

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

Table 3.2.: Multiset Natural Deduction version of STA.

where $\mathfrak{A}$ is a context, $\mathtt{M}$ is a λ-term and $\sigma \in \mathcal{T}$ is a soft type as defined in Definition 4. The rules defining $\mathrm{STA}_{\mathrm{M}}$ are depicted in Table 3.2.

$\mathrm{STA}_{\mathrm{M}}$ contexts are finite multiset of type assignments of the shape $\mathtt{x} : \sigma$, such that if $\mathtt{x} : \sigma_1 \in \mathfrak{A}$ and $\mathtt{x} : \sigma_2 \in \mathfrak{A}$ then there exist $A \in \mathcal{T}$ and $n, m \in \mathbb{N}$ such that $\sigma_1 \equiv !^n A$ and $\sigma_2 \equiv !^m A$. Contexts are ranged over by $\mathfrak{A}$,$\mathfrak{B}$,$\mathfrak{C}$. As usual $\mathfrak{A} \subseteq \mathfrak{B}$ denotes the fact that $\mathtt{x} : \sigma \in \mathfrak{A}$ with multiplicity n implies $\mathtt{x} : \sigma \in \mathfrak{B}$ with multiplicity $m \geq n$.

Moreover, $\mathfrak{A}, \mathfrak{B}$ denotes multiset union, i.e. the union with the sum of multiplicities. Note that in general, the union of two contexts $\mathfrak{A}$ and $\mathfrak{B}$ is not a context itself, in fact it can be the case that $\mathtt{x} : \sigma_1 \in \mathfrak{A}$ and $\mathtt{x} : \sigma_2 \in \mathfrak{B}$ and there is no $A \in \mathcal{T}$, $n, m \in \mathbb{N}$ such that $\sigma_1 \equiv !^n A$ and $\sigma_2 \equiv !^m A$. Two contexts $\mathfrak{A}$ and $\mathfrak{B}$ are *coherent*, denoted $\mathfrak{A} \approx \mathfrak{B}$, if and only if their union $\mathfrak{A}, \mathfrak{B}$ is a context. The notations introduced in Chapter 2 for set contexts can be easily extended to multiset contexts, e.g. $\mathrm{dom}(\mathfrak{A})$, $\mathrm{FV}(\mathfrak{A})$, $!\mathfrak{A}$, $\mathrm{FTV}(\mathfrak{A})$. Clearly a multiset of type assignments can in particular be a set of type assignments. We denote as usual set of type assignments as Γ, Δ. For $\mathrm{STA}_{\mathrm{M}}$ derivations we will use the same notation used in Chapter 2 for STA derivations. We hope that it will be clear from the context which system we refer to.

Some comments on the rules of the system follow. As expected, in rule $(\multimap E)$ there is the side condition $\mathfrak{A} \approx \mathfrak{B}$ assuring that the union of the contexts $\mathfrak{A}$ and $\mathfrak{B}$ is well defined. In the rule (m) no renaming occurs. Moreover, it always permits to collapse a multiset context into a set context.

The last remark can be made more precise. In fact we can define a maps from multiset contexts to set contexts.

Definition 13. *Let* $[\ ]^{\#}$ *be the mapping between* $\mathrm{STA}_{\mathrm{M}}$ *contexts inductively defined as:*

$$[\ \emptyset]^{\#} = \emptyset \qquad [\mathfrak{A}, \mathtt{x} : \sigma]^{\#} = [\mathfrak{A}]^{\#}, \mathtt{x} : \sigma \quad \textit{if } \mathtt{x} \notin \mathrm{dom}(\mathfrak{A})$$

$$[\mathfrak{A}, \mathtt{x} :!^{n_1} A, \ldots, \mathtt{x} :!^{n_k} A]^{\#} = [\mathfrak{A}]^{\#}, \mathtt{x} :!^{m+1} A \quad \textit{if } k > 1,\ m = \max_{j=1}^{k} n_j \textit{ and } \mathtt{x} \notin \mathrm{dom}\mathfrak{A}$$

Then we clearly have the following lemma.

Lemma 39.

$$\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma \textit{ implies } [\mathfrak{A}]^{\#} \vdash_{\mathrm{M}} \mathtt{M} : \sigma$$

Proof. Easy, by repeatedly applying (m) rule. □

In what follows, as in the case of STA, we will need to talk about proofs modulo commutations of rules.

Definition 14. *Let* Π *and* Π' *be two derivations in* $\mathrm{STA}_{\mathrm{M}}$*, proving the same conclusion:* $\Pi \rightsquigarrow \Pi'$ *denotes the fact that* Π' *is obtained from* Π *by commuting or deleting some rules.*

As usual the Generation Lemma connects the shape of a term with its typing.

Lemma 40 (Generation lemma).

1. $\Pi \rhd \mathfrak{A} \vdash_{\mathrm{M}} \lambda \mathtt{y}.\mathtt{M} : \forall\alpha.A$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$ *where the last rule of* Π' *is* $(\forall I)$.
2. $\Pi \rhd \mathfrak{A} \vdash_{\mathrm{M}} \lambda \mathtt{y}.\mathtt{M} : \sigma \multimap A$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* $(\multimap I)$.
3. $\Pi \rhd \mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : !\sigma$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$ *where* Π' *consists of a subderivation, ending with the rule (sp) proving* $!\mathfrak{A}' \vdash_{\mathrm{M}} \mathtt{M} : !\sigma$*, followed by a (maybe empty) sequence* δ *of rules (w) and/or (m), dealing with variables not occurring in* M*.*
4. $\Pi \rhd !\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : !\sigma$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* (sp).

Proof. Similar to the proof of Generation Lemma 8. □

Moreover, in what follows we need the following analogous of Lemma 6.1.

Lemma 41. $\Pi \rhd \mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma$ *implies there exists* Π' *such that* $\Pi' \rhd \mathfrak{A}[\vec{A}/\vec{\alpha}] \vdash_{\mathrm{M}} \mathtt{M} : \sigma[\vec{A}/\vec{\alpha}]$.

Proof. Similar to the proof of Lemma 6.1. Note that the derivation Π' can be obtained through a rule by rule correspondence. □

The other main properties of $\mathrm{STA}_{\mathrm{M}}$ will be directly inherited by $\mathrm{STA}_{\mathrm{N}}$ once the equivalence between the two systems will be proved.

3.3.2. $\mathrm{STA}_{\mathrm{N}}$ vs $\mathrm{STA}_{\mathrm{M}}$

We would now prove that if a term is typable in $\mathrm{STA}_{\mathrm{N}}$ then it is also typable in $\mathrm{STA}_{\mathrm{M}}$. This proves one direction of the equivalence between $\mathrm{STA}_{\mathrm{M}}$ and $\mathrm{STA}_{\mathrm{N}}$. In fact what we show is that there is a rule by rule correspondence. This means that measure and complexity results given over $\mathrm{STA}_{\mathrm{N}}$ derivations holds in particular for the corresponding $\mathrm{STA}_{\mathrm{M}}$ derivations.
Firstly, we need to extend the notion of substitution to derivations. Let $\mathfrak{A}$ be a $\mathrm{STA}_{\mathrm{M}}$

context. $\mathfrak{A}[\mathtt{x}/\mathtt{y}]$ denotes the context obtained from $\mathfrak{A}$ by replacing every type assignment $\mathtt{y} : \sigma \in \mathfrak{A}$ by the type assignment $\mathtt{x} : \sigma$. Let Π be a derivation in $\mathrm{STA}_{\mathrm{M}}$ then $\Pi[\mathtt{x}/\mathtt{y}]$ is the derivation obtained by replacing every sequent $\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma$ in Π by the sequent $\mathfrak{A}[\mathtt{x}/\mathtt{y}] \vdash_{\mathrm{M}} \mathtt{M}[\mathtt{x}/\mathtt{y}] : \sigma$. Note that substitution over contexts can transform a set in a multiset.
Now we can define a function mapping $\mathrm{STA}_{\mathrm{N}}$ derivation in $\mathrm{STA}_{\mathrm{M}}$ derivation.

Definition 15. *The function* $(\ )^*$ *mapping a* $\mathrm{STA}_{\mathrm{N}}$ *derivation* Π *in a* $\mathrm{STA}_{\mathrm{M}}$ *derivation* $(\Pi)^*$ *is defined inductively as:*

- *If* Π *end as:*
$$\frac{}{\mathtt{x} : A \vdash_{\mathrm{N}} \mathtt{x} : A}\ (Ax)$$
then:
$$(\Pi)^* = \frac{}{\mathtt{x} : A \vdash_{\mathrm{M}} \mathtt{x} : A}\ (Ax)$$
- *If* Π *end as:*
$$\frac{\Sigma \rhd \Gamma' \vdash_{\mathrm{N}} \mathtt{M}' : \sigma'}{\Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma}\ (R)$$
where R *is either a rule* $(w), (sp), (\multimap I), (\forall E)$ *or* $(\forall I)$*, then:*
$$(\Pi)^* = \frac{(\Sigma \rhd \Gamma' \vdash_{\mathrm{N}} \mathtt{M}' : \sigma')^*}{\Gamma \vdash_{\mathrm{M}} \mathtt{M} : \sigma}\ (R)$$
- *If* Π *end as:*
$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma \multimap A \quad \Theta \rhd \Delta \vdash_{\mathrm{N}} \mathtt{N} : \sigma}{\Gamma, \Delta \vdash_{\mathrm{N}} \mathtt{MN} : A}\ (\multimap E)$$
then:
$$(\Pi)^* = \frac{(\Sigma \rhd \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma \multimap A)^* \quad (\Theta \rhd \Delta \vdash_{\mathrm{N}} \mathtt{N} : \sigma)^*}{\Gamma, \Delta \vdash_{\mathrm{N}} \mathtt{MN} : A}\ (\multimap E)$$
- *If* Π *end as:*
$$\frac{\Sigma \rhd \Gamma, \mathtt{x}_1 : \tau, \ldots, \mathtt{x}_n : \tau \vdash_{\mathrm{N}} \mathtt{M} : \sigma}{\Gamma, \mathtt{x} : !\tau \vdash_{\mathrm{N}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)$$
then:
$$(\Pi)^* = \frac{(\Sigma \rhd \Gamma, \mathtt{x}_1 : \tau, \ldots, \mathtt{x}_n : \tau \vdash_{\mathrm{N}} \mathtt{M} : \sigma)^*[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n]}{\Gamma, \mathtt{x} : !\tau \vdash_{\mathrm{N}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)$$

So we can prove one direction of the equivalence.

Theorem 14. *Let* $\Pi \rhd \Gamma \vdash_{\mathrm{N}} \mathtt{M} : \sigma$*. Then there exists a derivation* $\Pi' \rhd \Gamma \vdash_{\mathrm{M}} \mathtt{M} : \sigma$*.*

Proof. Easy by induction on Π, taking $\Pi = (\Pi')^*$. □

3.3.3. A List natural deduction version of STA

In order to prove that for every $\mathrm{STA}_{\mathrm{M}}$ derivation there exists an analogous in $\mathrm{STA}_{\mathrm{N}}$, we introduce a further intermediary system $\mathrm{STA}_{\mathrm{S}}$. $\mathrm{STA}_{\mathrm{S}}$ is obtained starting by $\mathrm{STA}_{\mathrm{M}}$ and imposing an order between multiset elements. This permits to distinguish between different occurrences of the same element.
$\mathrm{STA}_{\mathrm{S}}$ proves sequents of the shape:

$$\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma$$

where $\mathcal{A}$ is a context, $\mathtt{M}$ is a λ-term and $\sigma \in \mathcal{T}$ is a soft type as defined in Definition 4. The rules defining $\mathrm{STA}_{\mathrm{S}}$ are depicted in Table 3.3.
$\mathrm{STA}_{\mathrm{S}}$ contexts are lists of type assignments of the shape $\mathtt{x} : \sigma$ such that if $\mathtt{x} : \sigma_1$ is in $\mathcal{A}$ and $\mathtt{x} : \sigma_2$ is in $\mathcal{A}$ then there exist $A \in \mathcal{T}$ and $n, m \in \mathbb{N}$ such that $\sigma_1 \equiv !^n A$ and $\sigma_2 \equiv !^m A$. Lists are denoted by objects of the shape:

$$[1 \leftarrowtail \langle \mathtt{x}_1 : \tau_1 \rangle, \ldots, n \leftarrowtail \langle \mathtt{x}_n : \tau_n \rangle]$$

Elements in a list are identified by their index, i.e. $\mathcal{A}[i]$ denotes the type assignment at position i. Index can have no assignment associated, i.e. $\mathcal{A}[i \leftarrowtail \epsilon]$ denotes that no assignment is associated to the index i. The length of a list $\mathcal{A}$, denoted $|\mathcal{A}|$ is the number of element of the list.
Contexts are ranged over by $\mathcal{A}$,$\mathcal{B}$. If $\mathcal{A} = [1 \leftarrowtail \langle \mathtt{x}_1 : \tau_1 \rangle, \ldots, n \leftarrowtail \langle \mathtt{x}_n : \tau_n \rangle]$ and $\mathcal{B} = [1 \leftarrowtail \langle \mathtt{y}_1 : \sigma_1 \rangle, \ldots, m \leftarrowtail \langle \mathtt{y}_m : \sigma_n \rangle]$ then the list concatenation of $\mathcal{A}$ and $\mathcal{B}$, denoted $\mathcal{A}@\mathcal{B}$, is the list:

$$\mathcal{A}@\mathcal{B} = [1 \leftarrowtail \langle \mathtt{x}_1 : \tau_1 \rangle, \ldots, n \leftarrowtail \langle \mathtt{x}_n : \tau_n \rangle, n+1 \leftarrowtail \langle \mathtt{y}_1 : \sigma_1 \rangle, \ldots, n+m \leftarrowtail \langle \mathtt{y}_m : \sigma_n \rangle]$$

Two list contexts $\mathcal{A}$ and $\mathcal{B}$ are *coherent*, denoted $\mathcal{A} \approx \mathcal{B}$, if and only if their concatenation $\mathcal{A}@\mathcal{B}$ is a list. Usually we identify a list of the shape $[1 \leftarrowtail \langle \mathtt{x} : \sigma \rangle]$ with the type assignment $\langle \mathtt{x} : \sigma \rangle$. Moreover, we use $\mathcal{A}[i_1 \leftarrowtail \langle \mathtt{x}_1 : \sigma_1 \rangle, \ldots, i_n \leftarrowtail \mathtt{x}_n : \sigma_n]$ to denote the contexts $\mathcal{A}$ where to the index i_j is assigned the type assignment $\langle \mathtt{x}_j : \sigma_j \rangle$ for $1 \leq j \leq n$.
The notations introduced in Chapter 2 for set contexts can be easily extended to list contexts, e.g. $\mathrm{dom}(\mathcal{A})$, $\mathrm{FV}(\mathcal{A})$, $!\mathcal{A}, \mathrm{FTV}(\mathcal{A})$. Furthermore, for $\mathrm{STA}_{\mathrm{S}}$ derivations we use the same notation introduced in Chapter 2 for STA derivations. We hope that it will be clear from the context which system we refer to.
List contexts can be easily identified with multiset contexts by using a map $\{\ \}^*$ forgetting the order structure. Moreover, it can be the case that a list context $\mathcal{A}$ is such that $\{\mathcal{A}\}^*$ is a set. As usual we write Γ, Δ for set of type assignments.
We here sketch a proof of the equivalence between $\mathrm{STA}_{\mathrm{S}}$ and $\mathrm{STA}_{\mathrm{M}}$.

Axiom and Weakening:

$$\frac{}{\langle \mathtt{x} : A \rangle \vdash_{\mathrm{S}} \mathtt{x} : A}\ (Ax) \qquad \frac{\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma}{\mathcal{A} @ \langle \mathtt{x} : A \rangle \vdash_{\mathrm{S}} \mathtt{M} : \sigma}\ (w)$$

Multiplicatives:

$$\frac{\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma \multimap A \quad \mathcal{B} \vdash_{\mathrm{S}} \mathtt{N} : \sigma \quad \mathcal{A} \approx \mathcal{B}}{\mathcal{A} @ \mathcal{B} \vdash_{\mathrm{S}} \mathtt{MN} : A}\ (\multimap E)$$

$$\frac{\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : A \quad \mathcal{A}[i] = \langle \mathtt{x} : \sigma \rangle \quad \mathcal{A}' = \mathcal{A}[i \hookleftarrow \epsilon] \quad \mathtt{x} \notin \mathrm{dom}(\mathcal{A}')}{\mathcal{A}' \vdash_{\mathrm{S}} \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap I)$$

Exponentials:

$$\frac{\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma \quad \mathcal{A}[i_1] = \cdots = \mathcal{A}[i_n] = \langle \mathtt{x} : \tau \rangle}{\mathcal{A}[i_1 \hookleftarrow \langle \mathtt{x} : !\tau \rangle, i_2 \hookleftarrow \epsilon, \ldots, i_n \hookleftarrow \epsilon] \vdash_{\mathrm{S}} \mathtt{M} : \sigma}\ (m)$$

$$\frac{\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma}{!\mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : !\sigma}\ (sp)$$

Quantifier:

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

Table 3.3.: List Natural Deduction version of STA.

Theorem 15.

$$\Pi \rhd \{\mathcal{A}\}^* \vdash_{\mathrm{M}} \mathtt{M} : \sigma \iff \Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma$$

Proof. Easy. Note that neither the subject nor the types change. The if part follows by induction on the derivation Π imposing a list structure to the multisets. The converse follows by induction on the derivation Σ by forgetting the list structure. □

3.3.4. $\mathrm{STA}_{\mathrm{M}}$ vs $\mathrm{STA}_{\mathrm{N}}$

In the sequel we need to talk of variables occurrences in a term. We identify a variable occurrence with the position in the syntactic tree of a term occurrence, e.g. the left occurrence of $\mathtt{x}$ in $\mathtt{xyx}$ is in the position 00 while the right one is in the position 1 and $\mathtt{y}$ is in the position ϵ in $\lambda\mathtt{x}.\mathtt{y}$.
The following definition connects a variable in a context with its occurrences in the subject.

Definition 16. *Let $\Pi \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma$. For every $i \leq |\mathcal{A}|$, $\mathrm{set}_i(\Pi \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma)$ is the set of variable occurrences in $\mathtt{M}$ associated to the variable at the i-th position in $\mathcal{A}$. $\mathrm{set}_i(\Pi \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma)$ is inductively defined on the structure of Π as follows:*

- *If the last applied rule of Π is:*

$$\frac{}{\langle \mathtt{x} : A \rangle \vdash_{\mathrm{S}} \mathtt{x} : A}\ (Ax)$$

  then $\mathrm{set}_1(\Pi) = \{\epsilon\}$.

- *If the last applied rule is:*

$$\frac{\Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma}{\mathcal{A} @ \langle \mathtt{x} : A \rangle \vdash_{\mathrm{S}} \mathtt{M} : \sigma}\ (w)$$

  then $\mathrm{set}_i(\Pi) = \mathrm{set}_i(\Sigma)$ for $1 \leq i \leq |\mathcal{A}|$ and $\mathrm{set}_{|\mathcal{A}|+1}(\Pi) = \emptyset$.

- *If the last applied rule is:*

$$\frac{\Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma \multimap A \quad \Theta \rhd \mathcal{B} \vdash_{\mathrm{S}} \mathtt{N} : \sigma \quad \mathcal{A} \approx \mathcal{B}}{\mathcal{A} @ \mathcal{B} \vdash_{\mathrm{S}} \mathtt{MN} : A}\ (\multimap E)$$

  then $\mathrm{set}_i(\Pi) = \{0i_k | i_k \in \mathrm{set}_i(\Sigma)\}$ for $1 \leq i \leq |\mathcal{A}|$ while $\mathrm{set}_i(\Pi) = \{1i_k | i_k \in \mathrm{set}_{i-|\mathcal{A}|}(\Theta)\}$ for $|\mathcal{A} + 1| \leq i \leq |\mathcal{A}| + |\mathcal{B}|$.

- *If the last applied rule is:*

$$\frac{\Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : A \quad \mathcal{A}[i] = \langle \mathtt{x} : \sigma \rangle \quad \mathcal{A}' = \mathcal{A}[i \leftarrow \epsilon] \quad \mathtt{x} \notin \mathrm{dom}(\mathcal{A}')}{\mathcal{A}' \vdash_{\mathrm{S}} \lambda\mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap I)$$

  then $\mathrm{set}_i(\Pi) = \emptyset$ and $\mathrm{set}_j(\Pi) = \mathrm{set}_j(\Sigma)$ for $1 \leq j \leq |\mathcal{A}|$ and $j \neq i$.

- *If the last applied rule is:*

$$\frac{\Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma \quad \mathcal{A}[i_1] = \cdots = \mathcal{A}[i_n] = \langle \mathtt{x} : \tau \rangle}{\mathcal{A}[i_1 \hookleftarrow \langle \mathtt{x} : !\tau \rangle, i_2 \hookleftarrow \epsilon, \ldots, i_n \hookleftarrow \epsilon] \vdash_{\mathrm{S}} \mathtt{M} : \sigma}\ (m)$$

  then for each $1 \leq k \leq |\mathcal{A}|$ *different from* $i_1, \ldots, i_n$*:* $\mathrm{set}_k(\Pi) = \mathrm{set}_k(\Sigma)$*. Moreover,* $\mathrm{set}_{i_1}(\Pi) = \bigcup_{j=0}^{n} \mathrm{set}_{i_j}(\Sigma)$ *and for each* $2 \leq j \leq n$*:* $\mathrm{set}_{i_j}(\Pi) = \emptyset$*.*

- *In every other case* $\mathrm{set}_i(\Pi)$ *is equal to* $\mathrm{set}_i(\Sigma)$ *where* Σ *is the only premise.*

Let $\Pi \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma$, then $\mathtt{M}\{\mathtt{x}_1/\mathrm{set}_1(\Pi), \cdots, \mathtt{x}_n/\mathrm{set}_n(\Pi)\}$ denotes the substitution of all the occurrences associated to the variable in the i-th position in $\mathcal{A}$ by the variable $\mathtt{x}_i$ for $1 \leq i \leq n$.
In order to prove the equivalence between $\mathrm{STA}_{\mathrm{N}}$ and $\mathrm{STA}_{\mathrm{M}}$ we need to partially rename variables. This can be done in $\mathrm{STA}_{\mathrm{S}}$, as shown by the following lemma.

Lemma 42. *Let* $\Pi \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma$*. Then, if* $\mathcal{A}[i_1] = \cdots = \mathcal{A}[i_n] = \langle \mathtt{x} : \tau \rangle$ *and* $\mathcal{A}' = \mathcal{A}[i_1 \hookleftarrow \langle \mathtt{x}_1 : \tau \rangle, \ldots, i_n \hookleftarrow \langle \mathtt{x}_n : \tau \rangle]$ *then:*

$$\mathcal{A}' \vdash_{\mathrm{S}} \mathtt{M}\{\mathtt{x}_1/\mathrm{set}_{i_1}(\Pi), \cdots, \mathtt{x}_n/\mathrm{set}_{i_n}(\Pi)\} : \sigma$$

Proof. By induction on Π. The base case is trivial. Consider the case Π ends as:

$$\frac{\Sigma \rhd \mathcal{B}_1 \vdash_{\mathrm{S}} \mathtt{N} : \sigma \multimap A \quad \Theta \rhd \mathcal{B}_2 \vdash_{\mathrm{S}} \mathtt{P} : \sigma \quad \mathcal{B}_1 \approx \mathcal{B}_2}{\mathcal{B}_1 @ \mathcal{B}_2 \vdash_{\mathrm{S}} \mathtt{NP} : A}\ (\multimap E)$$

and suppose $\mathcal{B}_1 @ \mathcal{B}_2[i_1] = \cdots = \mathcal{B}_1 @ \mathcal{B}_2[i_n] = \langle \mathtt{x} : \tau \rangle$. Then for some $j_1, \ldots, j_m$ and $k_1, \ldots, k_r$ we have $\mathcal{B}_1[j_1] = \cdots = \mathcal{B}_1[j_m] = \langle \mathtt{x} : \tau \rangle$ and $\mathcal{B}_2[k_1] = \cdots = \mathcal{B}_2[k_r] = \langle \mathtt{x} : \tau \rangle$ respectively. Let $\mathcal{B}_1' = \mathcal{B}_1[j_1 \hookleftarrow \langle \mathtt{x}_{j_1} : \tau \rangle, \cdots, j_m \hookleftarrow \langle \mathtt{x}_{j_m} : \tau \rangle]$ and $\mathcal{B}_2' = \mathcal{B}_2[k_1 \hookleftarrow \langle \mathtt{x}_{k_1} : \tau \rangle, \cdots, k_r \hookleftarrow \langle \mathtt{x}_{k_r} : \tau \rangle]$. Then, by induction hypothesis we have:

$$\Sigma' \rhd \mathcal{B}_1' \vdash_{\mathrm{S}} \mathtt{N}\{\mathtt{x}_{j_1}/\mathrm{set}_{j_1}(\Pi), \cdots, \mathtt{x}_{j_m}/\mathrm{set}_{j_m}(\Pi)\} : \sigma$$

and

$$\Theta' \rhd \mathcal{B}_2' \vdash_{\mathrm{S}} \mathtt{P}\{\mathtt{x}_{k_1}/\mathrm{set}_{k_1}(\Pi), \cdots, \mathtt{x}_{k_r}/\mathrm{set}_{k_r}(\Pi)\} : \sigma$$

clearly since $\mathcal{B}_1 \approx \mathcal{B}_2$ then also $\mathcal{B}_1' \approx \mathcal{B}_2'$, hence the conclusion follows by an application of the $(\multimap E)$ rule. Consider now the case $\tau \equiv !\tau'$, $j_1 = i_k$ for some $1 \leq k \leq n$ and Π ends as:

$$\frac{\Sigma \rhd \mathcal{A} \vdash_{\mathrm{S}} \mathtt{M} : \sigma \quad \mathcal{A}[j_1] = \cdots = \mathcal{A}[j_m] = \langle \mathtt{x} : \tau' \rangle}{\mathcal{A}[j_1 \hookleftarrow \langle \mathtt{x} : !\tau' \rangle, j_2 \hookleftarrow \epsilon, \ldots, j_m \hookleftarrow \epsilon] \vdash_{\mathrm{S}} \mathtt{M} : \sigma}\ (m)$$

Let

$$\mathcal{A}'' \equiv \mathcal{A}[i_1 \hookleftarrow \langle \mathtt{x}_1 : \tau \rangle, \ldots, i_{k-1} \hookleftarrow \langle \mathtt{x}_{k-1} : \tau \rangle, i_{k+1} \hookleftarrow \langle \mathtt{x}_{k+1} : \tau \rangle, \ldots, i_n \hookleftarrow \langle \mathtt{x}_n : \tau \rangle]$$

and let

$$\mathtt{M}' \equiv \mathtt{M}\{\mathtt{x}_1/\mathtt{set}_{i_1}(\Pi), \cdots, \mathtt{x}_{k-1}/\mathtt{set}_{i_{k-1}}(\Pi), \mathtt{x}_{k+1}\mathtt{set}_{i_{k+1}}(\Pi) \cdots, \mathtt{x}_n/\mathtt{set}_{i_n}(\Pi)\}$$

By induction hypothesis we have a derivation:

$$\Sigma' \rhd \mathcal{A}'' \vdash_{\mathtt{S}} \mathtt{M}' : \sigma$$

Since $\mathcal{A}''[j_1] = \cdots = \mathcal{A}''[j_m] = \langle \mathtt{x} : \tau' \rangle$ we can again apply induction hypothesis and obtain:

$$\mathcal{A}''[j_1 \leftarrow\!\shortmid \langle \mathtt{x}_k : \tau' \rangle, \ldots, j_m \leftarrow\!\shortmid \langle \mathtt{x}_k : \tau' \rangle] \vdash_{\mathtt{S}} \mathtt{M}\{\mathtt{x}_k/\mathtt{set}_{j_1}(\Sigma), \cdots, \mathtt{x}_k/\mathtt{set}_{j_m}(\Sigma)\}$$

then we can conclude:

$$\frac{\mathcal{A}''[j_1 \leftarrow\!\shortmid \langle \mathtt{x}_k : \tau' \rangle, \ldots, j_m \leftarrow\!\shortmid \langle \mathtt{x}_k : \tau' \rangle] \vdash_{\mathtt{S}} \mathtt{M}\{\mathtt{x}_k/\mathtt{set}_{j_1}(\Sigma), \cdots, \mathtt{x}_k/\mathtt{set}_{j_m}(\Sigma)\}}{\mathcal{A}''[j_1 \leftarrow\!\shortmid \langle \mathtt{x}_k : !\tau' \rangle, \ldots, j_m \leftarrow\!\shortmid \epsilon] \vdash_{\mathtt{S}} \mathtt{M}\{\mathtt{x}_k/\mathtt{set}_{j_1}(\Pi)\}}$$

The other cases follow directly by induction hypothesis. □

Now we can prove that typing in $\mathrm{STA}_{\mathtt{S}}$ corresponds to typing in $\mathrm{STA}_{\mathtt{N}}$.

Lemma 43. *Let $\mathcal{A}$ be a context such that $\{\mathcal{A}\}^* = \Gamma$ for some set Γ. Then:*

$$\mathcal{A} \vdash_{\mathtt{S}} \mathtt{M} : \sigma \implies \Gamma \vdash_{\mathtt{N}} \mathtt{M} : \sigma$$

Proof. By induction on the derivation Π proving $\mathcal{A} \vdash_{\mathtt{S}} \mathtt{M} : \sigma$. Base case is trivial. Let Π ends as:

$$\frac{\mathcal{A}_1 \vdash_{\mathtt{S}} \mathtt{N} : \sigma \multimap A \quad \mathcal{A}_2 \vdash_{\mathtt{S}} \mathtt{P} : \sigma \quad \mathcal{A}_1 \approx \mathcal{A}_2}{\mathcal{A}_1 @ \mathcal{A}_2 \vdash_{\mathtt{S}} \mathtt{NP} : A} \ (\multimap E)$$

Since $\{\mathcal{A}_1 @ \mathcal{A}_2\}^* = \Gamma$ for some set Γ, then clearly there are Γ_1 and Γ_2 such that $\{\mathcal{A}_1\}^* = \Gamma_1$, $\{\mathcal{A}_2\}^* = \Gamma_2$ and $\Gamma = \Gamma_1, \Gamma_2$. Hence by induction hypothesis we have $\Gamma_1 \vdash_{\mathtt{N}} \mathtt{N} : \sigma \multimap A$ and $\Gamma_2 \vdash_{\mathtt{N}} \mathtt{P} : \sigma$ and since clearly $\Gamma_1 \# \Gamma_2$, by an application of the rule $(\multimap E)$ we obtain $\Gamma_1, \Gamma_2 \vdash_{\mathtt{N}} \mathtt{NP} : A$. Hence, the conclusion follows.
Let Π end as:

$$\frac{\Pi' \rhd \mathcal{A} \vdash_{\mathtt{S}} \mathtt{M} : \sigma \quad \mathcal{A}[i_1] = \cdots = \mathcal{A}[i_n] = \langle \mathtt{x} : \tau \rangle}{\mathcal{A}[i_1 \leftarrow\!\shortmid \langle \mathtt{x} : !\tau \rangle, i_2 \leftarrow\!\shortmid \epsilon, \ldots, i_n \leftarrow\!\shortmid \epsilon] \vdash_{\mathtt{S}} \mathtt{M} : \sigma} \ (m)$$

and consider Π'. Clearly there exists a multiset $\mathfrak{A}$ such that $\{\mathcal{A}\}^* = \mathfrak{A}$ but it is not necessary a set. By Lemma 42 there exists Σ ending as:

$$\mathcal{A}[i_1 \leftarrow\!\shortmid \langle \mathtt{x}_1 : \tau \rangle, \ldots, i_n \leftarrow\!\shortmid \langle \mathtt{x}_n : \tau \rangle] \vdash_{\mathtt{S}} \mathtt{M}\{\mathtt{x}_1/\mathtt{set}_{i_1}(\Pi), \cdots, \mathtt{x}_n/\mathtt{set}_{i_n}(\Pi)\} : \sigma$$

In particular we can choose $\mathtt{x}_1, \ldots, \mathtt{x}_n$ fresh and distinct, hence for some set Γ':

$$\{\mathcal{A}[i_1 \leftarrowtail \langle \mathtt{x}_1 : \tau \rangle, \ldots, i_n \leftarrowtail \langle \mathtt{x}_n : \tau \rangle]\}^* = \Gamma'$$

By induction hypothesis we have $\Gamma' \vdash_{\mathrm{N}} \mathtt{M}\{\mathtt{x}_1/\mathtt{set}_{i_1}(\Pi), \cdots, \mathtt{x}_n/\mathtt{set}_{i_n}(\Pi)\}$ hence the conclusion follows by the next application of the (m) rule.

$$\frac{\Gamma' \vdash_{\mathrm{N}} \mathtt{M}\{\mathtt{x}_1/\mathtt{set}_{i_1}(\Pi), \cdots, \mathtt{x}_n/\mathtt{set}_{i_n}(\Pi)\} : \sigma}{\Gamma \vdash_{\mathrm{N}} \mathtt{M}\{\mathtt{x}_1/\mathtt{set}_{i_1}(\Pi), \cdots, \mathtt{x}_n/\mathtt{set}_{i_n}(\Pi)\}[\mathtt{x}/\mathtt{x}_1 \cdots \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)$$

The other cases follow directly by induction hypothesis. □

It is worth noting that the proof of the above lemma is in fact a rule by rule correspondence. Now we are ready to prove the other direction of the equivalence between the systems $\mathrm{STA}_{\mathrm{N}}$ and $\mathrm{STA}_{\mathrm{M}}$.

Theorem 16. *Let* $\Pi \rhd \mathfrak{A} \vdash_{\mathrm{M}} \mathtt{M} : \sigma$. *Then, there exists a derivation* $\Pi' \rhd [\mathfrak{A}]^{\#} \vdash_{\mathrm{N}} \mathtt{M} : \sigma$.

Proof. By Lemma 39, Theorem 15 and Lemma 43. □

4. A Taste of Type Inference for STA

4.1. Introduction

In this chapter we propose an algorithm $\mathbf{\Pi}$ for typing in the system $\mathrm{STA}_{\mathrm{M}}$ introduced in the previous chapter. For every λ-term, $\mathbf{\Pi}$ returns the set of constraints that need to be satisfied in order to type the term.
We start by defining a type containment relation and by introducing the notions of type schemes, of substitutions and of set of constraints. Then we give a unification algorithm that unifies type schemes and we show its correctness and completeness with respect to type schemes of a certain restricted form.
We prove that $\mathbf{\Pi}$ is correct in the sense that, given a term $\mathtt{M}$, if the constraints generated by $\mathbf{\Pi}(\mathtt{M})$ are satisfiable then there exists a typing for $\mathtt{M}$. Moreover, $\mathbf{\Pi}$ is also complete in the sense that, for every typable term $\mathtt{M}$, the set of constraints returned by $\mathbf{\Pi}(\mathtt{M})$ is satisfiable.
Satisfiability of constraints of the shape of the ones generated by the algorithm $\mathbf{\Pi}$ is in general undecidable. So, we ends this chapter by discussing a possible restriction of the system $\mathrm{STA}_{\mathrm{N}}$ for which satisfiability of constraints is decidable. Such restriction is interesting since it enjoys the same complexity properties of the entire system $\mathrm{STA}_{\mathrm{N}}$, notably (F)PTIME completeness.

4.2. Some preliminaries

In this section we introduce some notions which will be necessary in the sequel. Firstly we define a notion of type containment relation analogous to the one introduced in [Giannini and Ronchi Della Rocca, 1994]. Then, we introduce type schemes and we define scheme and ground substitutions mapping type schemes in type schemes and type schemes in types respectively. Finally we introduce the different kinds of constraints and what does it mean to satisfy them.

4.2.1. Type containment

We here adapt, to the case of soft types, the notion of containment relation used in [Giannini and Ronchi Della Rocca, 1994] for System F types. Recall that $\forall\vec{\alpha}.A$ is an abbreviation for $\forall\alpha_1 \ldots \forall\alpha_n.A$.

Definition 17. *The* containment relation $\leq_{\mathcal{T}}$ *between soft types is the relation defined as follows:*

$$\forall\vec{\alpha}.A \leq_{\mathcal{T}} A[\vec{B}/\vec{\alpha}]$$

The above containment relation is a slight modification of the one introduced in [Damas and Milner, 1982], [Mitchell, 1984] and [Mitchell, 1988].
The relation $\leq_{\mathcal{T}}$ is clearly decidable. Remembering that $\vec{\alpha}$ could be an empty sequence, $\leq_{\mathcal{T}}$ is obviously reflexive. Moreover, it is transitive, hence a preorder. Note that $\forall\vec{\alpha}.\tau \multimap \sigma \leq_{\mathcal{T}} \tau_1 \multimap \sigma_1$ implies $\forall\vec{\alpha}.\tau \leq_{\mathcal{T}} \tau_1$ and $\forall\vec{\alpha}.\sigma \leq_{\mathcal{T}} \sigma_1$, while in general the converse does not hold.
For what follows, the containment relation $\leq_{\mathcal{T}}$ introduced above suffices. Nevertheless, it is interesting to note that, following [Mitchell, 1988], $\leq_{\mathcal{T}}$ can be extended to a subtyping relation.

4.2.2. Schemes and substitutions

Definition 18. Linear type schemes *and* type schemes *are respectively defined by the grammars:*

$$U ::= a \mid \phi \multimap U \mid \forall t.a \mid \forall t.\phi \multimap U \quad \textit{(Linear type schemes)}$$
$$\phi ::= U \mid !^{p}U$$

where the exponential p *belongs to a set containing literals and the symbol* 0*,* a *belongs to a set of* linear scheme variables*,* t *belongs to a set containing* sequence variables *and the empty sequence* ε*.* $\mathcal{T}$ *denote the set of type schemes.*

Note that the symbol $\forall$ does not introduce bound variables. Moreover, note that schemes of the shape $\forall t.\forall u.U$ are not allowed.
$\mathrm{FV}(\phi)$ is the set of all linear scheme variables, sequence variables and exponentials occurring in ϕ. Linear scheme variables are ranged over by a, b, c, linear type schemes are ranged over by U, V, Z, type schemes are ranged over by ϕ, ψ, ξ, sequence variables are ranged over by t, u, v and exponentials are ranged over by p, q, r.
$\mathcal{T}^{-}$ denotes the set of non externally quantified linear types schemes, i.e. $\mathcal{T}^{-} = \{U \in \mathcal{T} \mid \text{for each } t, V : \ U \not\equiv \forall t.V\}$. A type scheme $!^{p}U$ is a modal type scheme. A type

scheme is *simple* if every variable or exponential appears in it at most once. Two type schemes ϕ, ψ are *distinct* if $\mathrm{FV}(\phi) \cap \mathrm{FV}(\psi) = \emptyset$
We need an equivalence over type schemes refining the syntactical one. $=^o \subseteq \mathcal{T} \times \mathcal{T}$ is the relation inductively defined as:

$$\phi =^\circ \psi \quad \text{if } \phi \equiv \psi \qquad\qquad \phi \multimap U =^\circ \psi \multimap V \quad \text{if } \phi =^\circ \psi \text{ and } U =^\circ V$$

$$\forall t.U =^\circ U \quad \text{if } t \equiv \varepsilon \qquad\qquad !^p\phi =^\circ \phi \quad \text{if } p \equiv 0$$

A *ground substitution* s is a total function mapping linear scheme variables to fixed non externally quantified linear types, sequence variables to (possibly empty) sequences of type variables, the symbol ϵ to empty sequence of type variables and exponentials to natural numbers. So a ground substitution maps type schemes to types. The application of s to a type scheme is inductively defined as:

$$s(a) = A \text{ if } [a \mapsto A] \in s \qquad s(\phi \multimap U) = s(\phi) \multimap s(U)$$

$$s(\forall t.U) = \begin{cases} s(U) & \text{if } [t \mapsto \varepsilon] \\ \forall \vec{\alpha}.s(U) & \text{if } [t \mapsto \vec{\alpha}] \end{cases} \qquad s(!^pU) =!^n s(U) \text{ if } [p \mapsto n] \in s$$

In what follows, $s[a_1 \mapsto \tau_1, \ldots, a_n \mapsto \tau_n]$ denotes the substitution defined as s except on variables $a_1, \ldots, a_n$ to which it assigns $\tau_1, \ldots \tau_n$.
A *scheme substitution* S is a total function mapping linear scheme variables to non externally quantified linear type schemes, sequence variables to sequence variables or the symbol ε and exponentials to exponentials or the symbol 0. Hence, a scheme substitution maps type schemes to type schemes. The application of S to a type scheme is inductively defined as:

$$S(a) = U \text{ if } [a \mapsto U] \in S, U \in \mathcal{T}^- \qquad S(\phi \multimap U) = S(\phi) \multimap S(U)$$

$$S(\forall t.U) = \forall t'.S(U) \text{ if } [t \mapsto t'] \in S \qquad S(!^pU) =!^q S(U) \text{ if } [p \mapsto q] \in S$$

In what follows, S_I denotes the identity substitution on schemes and $S[a_1 \mapsto U_1, \ldots, a_n \mapsto U_n]$ denotes the substitution defined as S except on variables $a_1, \ldots, a_n$ to which it assigns $U_1, \ldots U_n$.
A *type scheme context* is a multiset of *variable type scheme assignments* of the shape $\mathtt{x} : \phi$ where $\mathtt{x}$ is a variable and ϕ is a type scheme. Type scheme contexts are ranged over by Ψ, Φ. As usual, $\mathrm{dom}(\Psi)$ denotes the set $\{\mathtt{x} \mid \exists \mathtt{x} : \phi \in \Psi\}$, $\mathrm{range}(\Psi)$ denotes the set $\{\phi \mid \exists \mathtt{x} : \phi \in \Psi\}$, while $\mathrm{FV}(\Psi)$ denotes the set $\{\mathrm{FV}(\phi) \mid \exists \mathtt{x} : \phi \in \Psi\}$. Multiset union of type scheme contexts is denoted by $\sqcup$. The expression $\Phi = \Phi' \odot \Psi$

denotes that $\Phi = \Phi' \sqcup \Psi$ and $\mathrm{dom}(\Phi') \cap \mathrm{dom}(\Psi) = \emptyset$. Substitutions (either scheme or ground) are easily extended to type scheme contexts, *i.e.* if ρ is a substitution and $\Psi = \mathtt{x}_1 : \phi_1, \ldots, \mathtt{x}_n : \phi_n$ then $\rho(\Psi) = \mathtt{x}_1 : \rho(\phi_1), \ldots, \mathtt{x}_n : \rho(\phi_n)$.

4.2.3. Constraints and satisfiability

A *constraints sequence* $\mathcal{H}$ is a triple $\langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ of constraints sets. The set of *scheme variable constraints* $\mathcal{P}$ is a set of constraints of the shape $a = U$ where a is a linear scheme variable and U is a linear type scheme.
The set of *sequence variable constraints* $\mathcal{Q}$ is a set of constraints of the shape $t = t_1$ where t is a sequence variable and t_1 is either a sequence variable or the symbol ε. The set of *exponentials constraints* $\mathcal{C}$ is a set of linear constraints of the shape $p = q, p \geq q, p \geq q_1 + q_2, p > q, p = 0, p \geq 0$ or $p > 0$.
Let $\mathcal{H}_1 = \langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle$ and $\mathcal{H}_2 = \langle \mathcal{P}_2, \mathcal{Q}_2, \mathcal{C}_2 \rangle$ be two constraints sequences. $\mathcal{H}_1 \uplus \mathcal{H}_2$ denotes the constraints sequence $\langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle$. Sometimes we omit the empty set of a constraints sequence, i.e. $\mathcal{H} \uplus \{p = q\}$ denotes $\mathcal{H} \uplus \langle \emptyset, \emptyset, \{p = q\} \rangle$.
A *scheme system* $\mathcal{G}$ is a set of pairs of type schemes. A set of *binding constraints* $\mathcal{F}$ is a function from sequence variables to finite sets of schemes.

Definition 19.

i) If $a_1 \mathrm{op}_1 b_1, \ldots, a_n \mathrm{op}_n b_n$ *are constraints, where* $\mathrm{op}_1, \cdots, \mathrm{op}_n$ *are logical operators, then a substitution* ρ *(either scheme or ground) satisfies them if and only if* $\rho(a_1)\mathrm{op}_1\rho(b_1) = \cdots = \rho(a_n)\mathrm{op}_n\rho(b_n) = \mathtt{true}$

ii) Let S *be a scheme substitution.* S *satisfies a constraint sequence* $\mathcal{H} = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ *if and only if it satisfies the equality constraints of all the sets* $\mathcal{P}$, $\mathcal{Q}$ *and* $\mathcal{C}$.

iii) Let s *be a ground substitution.*

- s *satisfies a scheme system* $\mathcal{G} = \{(U_1, V_1), \ldots, (U_n, V_n)\}$ *if and only if* $s(U_1) \leq_{\mathcal{T}} s(V_1)$.
- s *satisfies a binding constraints* $\mathcal{F} = \{u_1 \mapsto \Gamma_1, \ldots, u_n \mapsto \Gamma_n\}$ *if and only if* $\forall i \leq n, \forall \alpha \in s(u_i), \forall U \in \Gamma_i : \alpha \notin \mathrm{FV}(s(U))$
- s *satisfies a constraints sequence* $\mathcal{H} = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$
  if and only if it satisfies all the exponential constraints $\mathcal{C}$.

The task of finding a scheme substitution satisfying a constraint sequence corresponds with minor modifications to first order unification. Giannini and Ronchi Della Rocca

$$\frac{}{\mathtt{U}(a, a) = \langle \emptyset, \emptyset, \emptyset \rangle}\ (\mathtt{U}_1) \qquad \frac{U \in \mathcal{T}^- \quad a \notin \mathrm{FV}(U)}{\mathtt{U}(a, U) = \langle \{a = U\}, \emptyset, \emptyset \rangle}\ (\mathtt{U}_2)$$

$$\frac{\mathtt{U}(a, U) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(a, \forall t.U) = \langle \mathcal{P}, \mathcal{Q} \cup \{t = \epsilon\}, \mathcal{C} \rangle}\ (\mathtt{U}_3) \qquad \frac{\mathtt{U}(a, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(a, !^p V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \cup \{p = 0\} \rangle}\ (\mathtt{U}_4)$$

$$\frac{\mathtt{U}(\phi, \psi) = \langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle \quad \mathtt{U}(U, V) = \langle \mathcal{P}_2, \mathcal{Q}_2, \mathcal{C}_2 \rangle}{\mathtt{U}(\phi \multimap U, \psi \multimap V) = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle}\ (\mathtt{U}_5)$$

$$\frac{\mathtt{U}(\phi \multimap U, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(\phi \multimap U, \forall t.V) = \langle \mathcal{P}, \mathcal{Q} \cup \{t = \varepsilon\}, \mathcal{C} \rangle}\ (\mathtt{U}_6) \qquad \frac{\mathtt{U}(\phi \multimap U, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(\phi \multimap U, !^p V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \cup \{p = 0\} \rangle}\ (\mathtt{U}_7)$$

$$\frac{\mathtt{U}(U, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(\forall t.U, \forall u.V) = \langle \mathcal{P}, \mathcal{Q} \cup \{t = u\}, \mathcal{C} \rangle}\ (\mathtt{U}_8) \qquad \frac{\mathtt{U}(\forall t.U, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(\forall t.U, !^p V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \cup \{p = 0\} \rangle}\ (\mathtt{U}_9)$$

$$\frac{\mathtt{U}(\psi, \phi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}\ (\mathtt{U}_{10}) \qquad \frac{\mathtt{U}(U, V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle}{\mathtt{U}(!^p U, !^q V) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \cup \{p = q\} \rangle}\ (\mathtt{U}_{11})$$

Table 4.1.: Unification Algorithm

in [Giannini and Ronchi Della Rocca, 1994] have shown how to effectively find partial solutions to scheme system and binding constraints.

4.3. Unification Algorithm

As usual in order to perform type inference we need a unification algorithm. We introduce an algorithm which permits to unify type schemes under some assumptions. The algorithm U is defined by the rules in Table 4.1. U proves judgments of the shape

$$\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$$

where ϕ and ψ are the two schemes that must be unified and $\langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ is a constraint sequence.

The unification algorithm in Table 4.1 is a simplified instance of Robinson unification algorithm. In fact, since the symbol $\forall$ does not introduce bounded variables in type schemes, we can consider it as a first order binary symbol. Then the unification problem we are considering is an instance of first order unification. Moreover in the sequel we

are interested in considering only type schemes of a particular shape.
Note that rule ($\mathtt{U}_{10}$) keeps down the number of rules nevertheless it can be cause of non termination (infinite derivations). It is easy to give a different definition of the algorithm dealing directly with symmetric rules. By now we will consider only finite derivations, hence we consider the algorithm as always terminating. Note that some inputs does not admit neither finite nor infinite derivations, in such a cases the unification fails.
In the sequel it will be sufficient the following weak form of correctness for the unification algorithm $\mathtt{U}$.

Theorem 17 ($\mathtt{U}$ Correctness). *Let $\phi, \psi \in \mathcal{T}$ be simple and distinct. If $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ then for every scheme substitution S satisfying $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{C}$:*

$$S(\phi) =^{\circ} S(\psi)$$

Proof. Firstly note that the unification algorithm $\mathtt{U}$ builds sequence variable constraints of the shape $t = u$ or $t = \varepsilon$. Since a substitution assigns sequence variables to sequence variables or to the symbol ε it is always possible to build a substitution satisfying these kind of constraints. Analogously, $\mathtt{U}$ builds exponential constraints of the shape $p = q$ or $p = 0$. Hence again it is always possible to build a substitution satisfying these kind of constraints. On the other hand, the case of scheme variable constraints is not so obvious. We prove the theorem by induction on the derivation with conclusion $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$. Since ϕ and ψ are distinct then the only possible base case is when the derivation is an application of the rule ($\mathtt{U}_2$), where $\phi \equiv a$ and $\psi \equiv U$ for some scheme variable a and $U \in \mathcal{T}^{-}$. Obviously, in such case S is every substitution assigning to a the scheme U.
Consider the case the derivation ends by an instance of the rule ($\mathtt{U}_{10}$), then the conclusion follows directly by induction hypothesis.
Let the derivation ends as:

$$\frac{\mathtt{U}(\phi', \psi') = \langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle \quad \mathtt{U}(U, V) = \langle \mathcal{P}_2, \mathcal{Q}_2, \mathcal{C}_2 \rangle}{\mathtt{U}(\phi' \multimap U, \psi' \multimap V) = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle} \ (\mathtt{U}_5)$$

By induction hypothesis we have S_1 satisfying $\langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle$ such that $S_1(\phi') =^{\circ} S_1(\psi')$ and S_2 satisfying $\langle \mathcal{P}_2, \mathcal{Q}_2, \mathcal{C}_2 \rangle$ such that $S_2(U) =^{\circ} S_2(V)$. Since $\phi' \multimap U$ and $\psi' \multimap V$ are simple and distinct it is easy to build a substitution S, satisfying both $\mathcal{Q}_1 \cup \mathcal{Q}_2$ and $\mathcal{C}_1 \cup \mathcal{C}_2$, which acts on ground variables in $\mathrm{FV}(\phi' \multimap U)$ as S_1 and on ground variables in $\mathrm{FV}(\psi' \multimap V)$ as S_2. Hence S satisfy $\langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle$ and $S(\phi' \multimap U) =^{\circ} S(\psi' \multimap V)$. So the conclusion follows. The other cases are similar. □

The completeness of the unification algorithm $\mathtt{U}$ is stated by the following theorem.

Theorem 18 (U Completeness). *If $S(\phi) =^{\circ} S(\psi)$ then $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ and S satisfies $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{C}$.*

Proof. By induction on the shape of ϕ and ψ. The base cases are obvious. So consider $\phi \equiv a$, $\psi \equiv \forall t.U$ and a substitution S such that $S(a) =^{\circ} S(\forall t.U)$. By definition of scheme substitution and of the equivalence $=^{\circ}$, S is such that $S(a) =^{\circ} S(U)$, hence $[t \mapsto \varepsilon] \in S$. By induction hypothesis we have $\mathtt{U}(a, U) = \langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle$ and S satisfies $\mathcal{P}_1$, $\mathcal{Q}_1$ and $\mathcal{C}_1$. Applying the rule $(\mathtt{U}_3)$ we obtain $\mathtt{U}(a, \forall t.U) = \langle \mathcal{P}_1, \mathcal{Q}_1 \cup \{t = \varepsilon\}, \mathcal{C}_1 \rangle$ and obviously S satisfies $\mathcal{P}_1$, $\mathcal{Q}_1 \cup \{t = \varepsilon\}$ and $\mathcal{C}_1$.
Consider the case $\phi \equiv \phi' \multimap U$, $\psi \equiv \psi' \multimap V$ and there is a substitution S such that $S(\phi' \multimap U) =^{\circ} S(\psi' \multimap V)$. By definition of scheme substitution and of the equivalence $=^{\circ}$, S is such that $S(\phi') =^{\circ} S(\psi')$ and $S(U) =^{\circ} S(V)$. By induction hypothesis $\mathtt{U}(\phi', \psi') = \langle \mathcal{P}_1, \mathcal{Q}_1, \mathcal{C}_1 \rangle$ and $\mathtt{U}(U, V) = \langle \mathcal{P}_2, \mathcal{Q}_2, \mathcal{C}_2 \rangle$, so by applying rule $(\mathtt{U}_5)$ it follows $\mathtt{U}(\phi' \multimap U, \psi' \multimap V) = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{C}_1 \cup \mathcal{C}_2 \rangle$. Moreover, S satisfies $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\mathcal{C}_1$ and $\mathcal{C}_2$, hence obviously it satisfies also $\mathcal{P}_1 \cup \mathcal{P}_2$, $\mathcal{Q}_1 \cup \mathcal{Q}_2$ and $\mathcal{C}_1 \cup \mathcal{C}_2$. So the conclusion follows. The other cases are similar. □

4.4. The Algorithm

The type inference algorithm defined in Table 4.2 proves statement of the shape:

$$\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, \phi, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$$

where Ψ is a type scheme assignment context, ϕ is a type scheme, $\mathcal{G}$ is a scheme system, $\mathcal{F}$ is a set of binding constraints and $\mathcal{H}$ is a constraints sequence.
The type inference algorithm recall the `Unify` procedure, defined in Table 4.3, on contexts and schemes which need to be unified through the unification algorithm.
The next Lemma assures that the scheme which need to be unified are distinct.

Lemma 44. *Let $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. If $\mathtt{x} : \phi \in \Psi$ then U and ϕ are simple and distinct. Moreover, if $\mathtt{x} : \phi, \mathtt{x} : \psi \in \Psi$ then ϕ and ψ are simple and distinct.*

Proof. By induction on $\mathtt{M}$. Base case is trivial. So let $\mathtt{M} \equiv \lambda \mathtt{x}.\mathtt{N}$ and $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, V, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$. Assume $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}} \odot \{\mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n\}$ for some $n \geq 0$. Clearly if $n = 0$ then the conclusion follows directly by the freshness condition. Otherwise if $n = 1$, by induction hypothesis for every $\mathtt{y} : \phi \in \Psi_N$ we have that V and ϕ are simple and distinct. So in particular $!^{s_1} V_1$ and V are distinct and simple, hence $U \equiv !^{s_1} V_1 \multimap V$ itself is simple. The case $n > 1$ is similar. Moreover, if $\mathtt{y} : \phi, y : \psi \in \Psi'_{\mathtt{N}}$ by induction

$$\mathbf{\Pi}(\mathtt{x}) = \mathtt{let}\ u,t,a,b,p\ \mathtt{be\ fresh\ in}$$
$$\langle \{\mathtt{x} :!^{p}\forall t.a\}, \forall u.b, \{(\forall t.a, b)\}, [u \mapsto \{\forall t.a\}], \{\emptyset, \emptyset, \{p \geq 0\}\}\rangle$$

$$\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{M}) = \mathtt{let}\ \mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H}\rangle\ \mathtt{in}$$
$$\mathtt{let}\ \Psi = \Psi' \odot \{\mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n\}\ , \mathcal{I} = \mathtt{range}(\Psi')\ \mathtt{in}$$
$$\mathtt{let}\ u,t,a,r\ \mathtt{be\ fresh\ in}$$
$$\mathtt{if}\ n = 0\ \mathtt{then}$$
$$\langle \Psi', \forall u.!^{r}(\forall t.a) \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \uplus \{r \geq 0\}\rangle$$
$$\mathtt{else\ if}\ n = 1\ \mathtt{then}$$
$$\langle \Psi', \forall u.!^{r} V_1 \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \uplus \{r \geq s_1, \}\rangle$$
$$\mathtt{else\ if}\ n > 1\ \mathtt{then}$$
$$\langle \Psi', \forall u.!^{r} V_1 \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \uplus \{r > s_1, \ldots, r > s_n\}\rangle$$

$$\mathbf{\Pi}(\mathtt{MN}) = \mathtt{let}\ \mathbf{\Pi}(\mathtt{M}) = \langle \Psi_{\mathtt{M}}, U, \mathcal{G}_{\mathtt{M}}, \mathcal{F}_{\mathtt{M}}, \mathcal{H}_{\mathtt{M}}\rangle\ \mathtt{and}$$
$$\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, V, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}}\rangle\ \ \mathtt{be\ distinct\ in}$$
$$\mathtt{let}\ u,t,a,b,q_i,p\ \mathtt{be\ fresh\ in}$$
$$\mathtt{let}\ \Psi'_{\mathtt{N}} = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_{\mathtt{N}}\}, \mathcal{I} = \mathtt{range}(\Psi_{\mathtt{M}} \sqcup \Psi_{\mathtt{N}})$$
$$\mathcal{H} = \mathtt{Unify}(\Psi_{\mathtt{M}}, \Psi'_{\mathtt{N}}, U, !^{p}V \multimap \forall t.a)\ \mathtt{in}$$
$$\langle \Psi_{\mathtt{M}} \sqcup \Psi'_{\mathtt{N}}, \forall u.b, \mathcal{G}_{\mathtt{M}} \cup \mathcal{G}_{\mathtt{N}} \cup \{(\forall t.a, b)\}, \mathcal{F}_{\mathtt{M}} + \mathcal{F}_{\mathtt{N}} + [u \mapsto \mathcal{I}],$$
$$\mathcal{H}_{\mathtt{M}} \uplus \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H} \uplus \{q_i \geq p_i + p\}\rangle$$

Table 4.2.: Type Inference Algorithm

$$
\begin{array}{rcl}
\mathtt{Unify}(\Phi, \Psi, \phi, \psi) & = & \mathtt{let}\ \mathtt{x}_1, \ldots, \mathtt{x}_m = \mathrm{dom}(\Phi) \cap \mathrm{dom}(\Psi),\ \forall 1 \leq i < m \\
& & \quad \Phi(\mathtt{x}_i) = \{!^{s_1} V_1, \ldots, !^{s_n} V_n\} \\
& & \quad \Psi(\mathtt{x}_i) = \{!^{r_1} U_1, \ldots, !^{r_k} U_k\}, \\
& & \quad \mathtt{U}(\phi, \psi) = \langle \mathcal{P}_0, \mathcal{Q}_0, \mathcal{C}_0 \rangle \\
& & \quad \mathtt{U}(V_1, U_1) = \langle \mathcal{P}_i, \mathcal{Q}_i, \mathcal{C}_i \rangle \\
& & \mathtt{in} \langle \bigcup_{j=0}^{m} \mathcal{P}_j, \bigcup_{j=0}^{m} \mathcal{Q}_j, \bigcup_{j=0}^{m} \mathcal{C}_j \rangle,
\end{array}
$$

Table 4.3.: `Unify` procedure

hypothesis it follows directly that ϕ and ψ are distinct and simple.
The case $\mathtt{M} \equiv \mathtt{PQ}$ is easier, it follows directly by induction hypothesis and freshness conditions. □

The above lemma is useful to prove that the type inference algorithm is well defined.

Theorem 19 ($\mathbf{\Pi}$ Termination). *For every $\mathtt{M} \in \Lambda$, there exist $\Psi, U, \mathcal{G}, \mathcal{F}$ and $\mathcal{H}$ such that:*

$$\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$$

Proof. By induction on the structure of $\mathtt{M}$. The base case is trivial. The case where $\mathtt{M} \equiv \lambda \mathtt{x}.\mathtt{N}$ follows directly by induction hypothesis. So, consider the case $\mathtt{M} \equiv \mathtt{N}_1\mathtt{N}_2$. By induction hypothesis we have $\mathbf{\Pi}(\mathtt{N}_1) = \langle \Psi_1, U_1, \mathcal{G}_1, \mathcal{F}_1, \mathcal{H}_1 \rangle$ and $\mathbf{\Pi}(\mathtt{N}_2) = \langle \Psi_2, U_2, \mathcal{G}_2, \mathcal{F}_2, \mathcal{H}_2 \rangle$. Let $\Psi_2' = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_2\}$. Then, it suffices to show that $\mathtt{Unify}(\Psi_1, \Psi_2', \mathtt{U}_1, !^{\mathtt{p}}\mathtt{U}_2 \multimap \forall \mathtt{t}.\mathtt{a})$ ends. By hypothesis and freshness condition U_1, $!^p U_2 \multimap \forall t.a$ and the schemes in Ψ_1 and Ψ_2' are all distinct. Moreover, by Lemma 44 they are simple. Hence, by Theorem 17 the procedure ends and so the conclusion follows. □

The following lemma justifies the fact that in the definition of $\mathbf{\Pi}$, in the abstraction case we only take the type scheme of the first occurrence (if any) of the variable to be abstracted. The same hold for the `Unify` procedure.

Lemma 45. *Let $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. Then, there exists a scheme substitution S satisfying $\mathcal{H}$ such that if $\mathtt{x} :!^{n_1} V_1, \mathtt{x} :!^{n_2} V_2 \in \Psi$ then $S(V_1) =^{\circ} S(V_2)$.*

Proof. By induction on $\mathbf{\Pi}(\mathtt{M})$ using Lemma 44 and Theorem 17 □

We can now finally prove the main theorems of this section. Firstly, we prove that the algorithm $\mathbf{\Pi}$ is correct with respect to the type assignment $\mathrm{STA}_{\mathtt{M}}$.

Theorem 20 ($\mathbf{\Pi}$ Correctness). *Let $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. Then, for each scheme substitution S satisfying $\mathcal{H}$, if there exists a ground substitution s satisfying $S(\mathcal{G})$, $S(\mathcal{F})$ and $S(\mathcal{H})$, then:*

$$s(S(\Psi)) \vdash_{\mathtt{M}} \mathtt{M} : s(S(U))$$

Proof. By induction on the derivation proving $\mathbf{\Pi}(\mathtt{M})$. The base case is obvious.
Consider the case $\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{N}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. By hypothesis $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$. Let S be a scheme substitution satisfying $\mathcal{H}$ and suppose there is a ground substitution s satisfying $S(\mathcal{G})$, $S(\mathcal{F})$ and $S(\mathcal{H})$ respectively. By induction hypothesis since S clearly satisfies $\mathcal{H}_{\mathtt{N}}$ and s satisfies $S(\mathcal{G})$, $S(\mathcal{F}_{\mathtt{N}})$ and $S(\mathcal{H}_{\mathtt{N}})$, we have:

$$s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))$$

Now, let $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}} \odot \{\mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n\}$. We have three distinct cases.
If $n = 0$ then $\mathtt{x} \notin \mathrm{dom}(\Psi_{\mathtt{N}})$ then by hypothesis $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}}$, $U \equiv \forall u.!^r(\forall t.a) \multimap U_{\mathtt{N}}$ and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \{r \geq 0\}$ for some fresh u, r, t, a. Hence in particular, there are $k \geq 0, \vec{\alpha}, \vec{\alpha}_1$ and A such that:

$$s(S(U)) \equiv s(S(\forall u.!^r(\forall t.a) \multimap U_{\mathtt{N}})) \equiv \forall \vec{\alpha}.!^k(\forall \alpha_1.A) \multimap s(S(U_{\mathtt{N}}))$$

Then, the conclusion follows by the next derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}{\mathtt{x} : \forall \vec{\alpha}_1.A, s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}\,(w)}{\mathtt{x} :!^k \forall \vec{\alpha}_1.A, s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}\,(m)^k}{s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} :!^k(\forall \vec{\alpha}_1.A) \multimap s(S(U_{\mathtt{N}}))}\,(\multimap I)}{s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \forall \vec{\alpha}.!^k(\forall \vec{\alpha}_1.A) \multimap s(S(U_{\mathtt{N}}))}\,(\forall I)^*$$

Note that we have freely applied the $(\forall I)$ rule over variables in $\vec{\alpha}$ since s satisfies the binding constraints $S(\mathcal{F})$.
If $n = 1$, then by hypothesis $U \equiv \forall u.!^r(V_1) \multimap U_{\mathtt{N}}$, $\Psi_{\mathtt{N}} = \mathtt{x} :!^{s_1} V_1, \Psi'_N$ and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \{r \geq s_1\}$ for some fresh r and u. By induction hypothesis we have $\mathtt{x} : s(S(!^{s_1} V_1)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))$. Since S satisfies $\mathcal{H}$ and s satisfies $S(\mathcal{G})$, $S(\mathcal{F})$ and $S(\mathcal{H})$, in particular there is $k \geq 0$ such that $s(S(!^r V_1)) \equiv !^k s(S(!^{s_1} V_1))$ and $\vec{\alpha}$ such that $s(S(u)) = \vec{\alpha}$. Then, the conclusion follows by the next derivation:

$$\dfrac{\dfrac{\dfrac{\mathtt{x} : s(S(!^{s_1} V_1)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}{\mathtt{x} :!^k s(S(!^{s_1} V_1)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}\,(m)^k}{s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} :!^k s(S(!^{s_1} V_1)) \multimap s(S(U_{\mathtt{N}}))}\,(\multimap I)}{s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \forall \vec{\alpha}.!^k s(S(!^{s_1} V_1)) \multimap s(S(U_{\mathtt{N}}))}\,(\forall I)^*$$

Again we have freely applied the $(\forall I)$ rule over variables in $\vec{\alpha}$ since s satisfies the binding constraints $S(\mathcal{F})$.
If $n > 1$ then by hypothesis $U \equiv \forall u.!^r(V_1) \multimap U_{\mathtt{N}}$, $\Psi_{\mathtt{N}} = \mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n$, $\Psi'_{\mathtt{N}}$ and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \{r > s_1, \ldots, r > s_n\}$ for fresh r, u. By induction hypothesis $\mathtt{x} : s(S(!^{s_1} V_1)), \ldots, \mathtt{x} : s(S(!^{s_n} V_n)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))$ and by Lemma 45 $s(S(V_1)) = \cdots = s(S(V_n))$. Since s satisfies $S(\mathcal{H})$ in particular there are $k_i \geq 0$ such that $s(S(r)) = s(S(s_i)) + k_i + 1$ for $1 \leq i \leq n$. Let $s(S(u)) = \vec{\alpha}$. Then, the conclusion follows by the next derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{x} : s(S(!^{s_1} V_1)), \ldots, \mathtt{x} : s(S(!^{s_n} V_n)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}{\mathtt{x} :!^{k_1} s(S(!^{s_1} V_1)), \ldots, \mathtt{x} :!^{k_n} s(S(!^{s_n} V_1)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}\,(m)^*}{\mathtt{x} :!^{k_1+1} s(S(!^{s_1} V_1)), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}\,(m)}{s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} :!^{k_1+1} s(S(!^{s_1} V_1)) \multimap s(S(U_{\mathtt{N}}))}\,(\multimap I)}{s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \forall \vec{\alpha}.!^{k_1+1} s(S(!^{s_1} V_1)) \multimap s(S(U_{\mathtt{N}}))}\,(\forall I)^*$$

Again we have freely applied the $(\forall I)$ rule over variables in $\vec{\alpha}$ since s satisfies the binding constraints $S(\mathcal{F})$.
Consider the case $\mathbf{\Pi}(\mathtt{MN}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. By hypothesis $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi_{\mathtt{M}}, U_{\mathtt{M}}, \mathcal{G}_{\mathtt{M}}, \mathcal{F}_{\mathtt{M}}, \mathcal{H}_{\mathtt{M}} \rangle$, $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ and $U \equiv \forall u.b$. Let S be a scheme substitution satisfying $\mathcal{H}$ and suppose there is a ground substitution s satisfying $S(\mathcal{G})$, $S(\mathcal{F})$ and $S(\mathcal{H})$ respectively. By induction hypothesis since S clearly satisfies $\mathcal{H}_{\mathtt{M}}$ and $\mathcal{H}_{\mathtt{N}}$ and moreover s satisfies $S(\mathcal{G}_{\mathtt{M}})$, $S(\mathcal{G}_{\mathtt{N}})$, $S(\mathcal{F}_{\mathtt{M}})$, $S(\mathcal{F}_{\mathtt{N}})$, $S(\mathcal{H}_{\mathtt{M}})$ and $S(\mathcal{H}_{\mathtt{N}})$, then we have both:

$$s(S(\Psi_{\mathtt{M}})) \vdash_{\mathtt{M}} \mathtt{M} : s(S(U_{\mathtt{M}})) \qquad \text{and} \qquad s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))$$

By hypothesis for t, a, p fresh: $\mathtt{U}(U_{\mathtt{M}}, !^p U_{\mathtt{N}} \multimap \forall t.a) = \mathcal{H}'$ with $\mathcal{H}' \subseteq \mathcal{H}$. Since $U_{\mathtt{M}}$ and $!^p U_{\mathtt{N}} \multimap \forall t.a$ are distinct and since S satisfies $\mathcal{H}$ by Theorem 17: $s(S(U_{\mathtt{M}})) \equiv s(S(!^p U_{\mathtt{N}})) \multimap s(S(\forall t.a))$. Let $\Psi'_{\mathtt{N}} = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_{\mathtt{N}}\}$. Then clearly $s(S(\Psi)) = s(S(\Psi_{\mathtt{M}} \sqcup \Psi'_{\mathtt{N}})) = s(S(\Psi_M)), s(S(\Psi'_{\mathtt{N}}))$. Moreover since by hypothesis s satisfies $S(\mathcal{G})$, then in particular $s(S(\forall t.a)) \leq s(S(b))$. So, let $s(S(u)) = \vec{\alpha}$ and $s(S(p)) = k$. Then, the conclusion follows by the next derivation:

$$\dfrac{\dfrac{\dfrac{s(S(\Psi_{\mathtt{M}})) \vdash_{\mathtt{M}} \mathtt{M} : s(S(!^p U_{\mathtt{N}})) \multimap s(S(\forall t.a)) \quad \dfrac{\dfrac{s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} : s(S(U_{\mathtt{N}}))}{!^k s(S(\Psi_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} :!^k s(S(U_{\mathtt{N}}))}\,(sp)^k}{s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{N} :!^k s(S(U_{\mathtt{N}}))}\,(m)^*}{s(S(\Psi_{\mathtt{M}})), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{MN} : s(S(\forall t.a))}\,(\multimap E)}{s(S(\Psi_{\mathtt{M}})), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{MN} : s(S(b))}\,(\forall E)^*}{s(S(\Psi_{\mathtt{M}})), s(S(\Psi'_{\mathtt{N}})) \vdash_{\mathtt{M}} \mathtt{MN} : \forall \vec{\alpha}.s(S(b))}\,(\forall I)^*$$

Again we have freely applied the $(\forall I)$ rule over variables in $\vec{\alpha}$ since s satisfies the binding constraints $S(\mathcal{F})$. □

Now we prove that the algorithm $\mathbf{\Pi}$ is also complete for the type assignment system $\mathrm{STA}_{\mathtt{M}}$.

Theorem 21 ($\mathbf{\Pi}$ Completeness). *Let $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. Then, if $\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{M} : \sigma$ then there exists a ground substitution s satisfying $\mathcal{G}, \mathcal{F}$ and $\mathcal{H}$ such that:*

$$\Sigma \rhd s(\Psi) \vdash_{\mathtt{M}} \mathtt{M} : s(U)$$

Moreover, the sequent $\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{M} : \sigma$ can be obtained from Σ by applying repeatedly (w), (m) and/or (sp) rules.

Proof. By induction on the derivation Π proving $\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{M} : \sigma$. The base case is when Π ends as:

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

Let $\mathbf{\Pi}(\mathtt{x}) = \langle \{\mathtt{x} :!^{p} \forall t.a\}, \forall u.b, \{(\forall t.a, b)\}, [u \mapsto \{\forall t.a\}], \{\emptyset, \emptyset, \{p \geq 0\}\} \rangle$ where a, b, t, u and p are fresh. Clearly for every ground substitution s' the substitution:

$$s = s'[a \mapsto A, b \mapsto B, t \mapsto \epsilon, u \mapsto \epsilon, p \mapsto 0]$$

satisfies the constraints. Hence, the conclusion follows trivially.
Consider the case where Π ends as:

$$\frac{\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathtt{M}} \mathtt{N} : A \quad \mathtt{x} \notin \mathrm{dom}(\mathfrak{A})}{\mathfrak{A} \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \sigma \multimap A}\ (\multimap I)$$

Let $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ where $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}} \odot \{\mathtt{x} :!^{r_1} V_1, \ldots, \mathtt{x} :!^{r_n} V_n\}$ and $\mathcal{I} = \mathrm{range}(\Psi'_{\mathtt{N}})$. Then, depending on the value of n, we have three distinct cases. If $n = 0$ then $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}}$ and for u, r, t and a fresh we have:

$$\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{N}) = \langle \Psi'_{\mathtt{N}}, \forall u.!^{r}(\forall t.a) \multimap U_{\mathtt{N}}, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \mapsto \mathcal{I}], \mathcal{H}_{\mathtt{N}} \uplus \{r \geq 0\} \rangle$$

By induction hypothesis there exists a ground substitution s satisfying $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{N}}$ such that:

$$\Sigma' \rhd s(\Psi_{\mathtt{N}}) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})$$

and $\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathtt{M}} \mathtt{N} : A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. Since $\mathtt{x} \notin \mathrm{FV}(s(\Psi_{\mathtt{N}}))$, in order to recover $\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathtt{M}} \mathtt{N} : A$ from Σ' it is necessary at least an application of the (w) rule, moreover A is linear hence no application of the (sp) rule occurs. Since u, r, t and a are fresh, we can extend s to a ground substitution s' such that $s'(u) \equiv \epsilon$ and $s'(!^{r} \forall t.a) \equiv \sigma$, hence in particular $s'(r) = k \geq 0$. Then:

$$s'(\forall u.!^{r}(\forall t.a) \multimap U_{\mathtt{N}}) \equiv s'(!^{r}(\forall t.a))) \multimap s(U_{\mathtt{N}}) \equiv \sigma \multimap A$$

and s' satisfies $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \to \mathcal{I}]$ and $\mathcal{H}_{\mathtt{N}} \uplus \{r \geq 0\}$. So Σ ends as follows:

$$\dfrac{\dfrac{\dfrac{\Sigma' \rhd s(\Psi_{\mathtt{N}}) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}{s(\Psi_{\mathtt{N}}), \mathtt{x} : s'(\forall t.a) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}\,(w)}{s(\Psi_{\mathtt{N}}), \mathtt{x} :!^k s'(\forall t.a) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}\,(m)^k}{s(\Psi_{\mathtt{N}}) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} :!^k s'(\forall t.a) \multimap s(U_{\mathtt{N}})}\,(\multimap I)$$

and clearly $\mathfrak{A} \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \sigma \multimap A$ can be obtained from it by applying repeatedly (m) and/or (w) rules.
If $n = 1$ then $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}}, \mathtt{x} :!^{r_1} V_1$ and for u and r fresh we have:

$$\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{N}) = \langle \Psi'_{\mathtt{N}}, \forall u.!^r V_1 \multimap U_{\mathtt{N}}, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \mapsto \mathcal{I}], \mathcal{H}_{\mathtt{N}} \uplus \{r \geq r_1\} \rangle$$

By induction hypothesis there exists a ground substitution s satisfying $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{N}}$ such that:

$$\Sigma' \rhd s(\Psi'_{\mathtt{N}}), \mathtt{x} : s(!^{r_1} V_1) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})$$

and $\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathtt{M}} \mathtt{N} : A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. Since u and r are fresh, we can extend s to a ground substitution s' such that $s'(u) \equiv \epsilon$ and $s'(!^r V_1) \equiv !^k s(!^{r_1} V_1) \equiv \sigma$, hence in particular $s'(r) = k + s(r_1)$ for some $k \geq 0$. Then:

$$s'(\forall u.!^r V_1 \multimap U_{\mathtt{N}}) \equiv !^k s(!^{r_1} V_1) \multimap s(U_{\mathtt{N}}) \equiv \sigma \multimap A$$

and s' satisfies $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \to \mathcal{I}]$ and $\mathcal{H}_{\mathtt{N}} \uplus \{r \geq r_1\}$. So, Σ ends as follows:

$$\dfrac{\dfrac{\Sigma' \rhd s(\Psi'_{\mathtt{N}}), \mathtt{x} : s(!^{r_1} V_1) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}{s(\Psi'_{\mathtt{N}}), \mathtt{x} :!^k s(!^{r_1} V_1) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}\,(m)^k}{s(\Psi'_{\mathtt{N}}) \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} :!^k s(!^{r_1} V_1) \multimap s(U_{\mathtt{N}})}\,(\multimap I)$$

and clearly $\mathfrak{A} \vdash_{\mathtt{M}} \lambda \mathtt{x}.\mathtt{N} : \sigma \multimap A$ can be obtained from it by applying repeatedly (m) and/or (w) rules.
If $n > 1$ then $\Psi_{\mathtt{N}} = \Psi'_{\mathtt{N}}, \mathtt{x} :!^{r_1} V_1, \ldots, \mathtt{x} :!^{r_n} V_n$ and for u and r fresh we have:

$$\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{N}) = \langle \Psi'_{\mathtt{N}}, \forall u.!^r V_1 \multimap U_{\mathtt{N}}, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \mapsto \mathcal{I}], \mathcal{H}_{\mathtt{N}} \uplus \{r > r_1, \ldots, r > r_n\} \rangle$$

By induction hypothesis there exists a ground substitution s satisfying $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{N}}$ such that:

$$\Sigma' \rhd s(\Psi'_{\mathtt{N}}), \mathtt{x} : s(!^{r_1} V_1), \ldots, \mathtt{x} : s(!^{r_n} V_n) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})$$

and $\mathfrak{A}, \mathtt{x} : \sigma \vdash_{\mathtt{M}} \mathtt{N} : A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. In particular, this implies that for every $1 \leq i \leq n$ there is a $k_i \geq 0$ such

that $!^{k_i+1}s(!^{r_i}V_i) \equiv \sigma$. Since u and r are fresh, we can extend s to a ground substitution s' such that $s'(u) = \epsilon$ and $s'(!^{r}V_1) \equiv !^{k_1+1}s(!^{r_1}V_1) \equiv \cdots \equiv !^{k_n+1}s(!^{r_n}V_n) \equiv \sigma$, hence in particular $s'(r) = k_1 + s(r_1) + 1 = \cdots = k_n + s(r_n) + 1$. Then:

$$s'(\forall u.!^{r}V_1 \multimap U_{\mathtt{N}}) \equiv !^{k_1+1}s(!^{r_1}V_1) \multimap s(U_{\mathtt{N}}) \equiv \sigma \multimap A$$

and s' satisfies $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}} + [u \to \mathcal{I}]$ and $\mathcal{H}_{\mathtt{N}} \uplus \{r \geq r_1, \ldots, r \geq r_n\}$. Then, Σ ends as follows:

$$\dfrac{\dfrac{\dfrac{s(\Psi'_{\mathtt{N}}), \mathtt{x} : s(!^{r_1}V_1), \ldots, \mathtt{x} : s(!^{r_n}V_n) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}{s(\Psi'_{\mathtt{N}}), \mathtt{x} :!^{k_1} s(!^{r_1}V_1), \ldots, \mathtt{x} :!^{k_n} s(!^{r_n}V_n) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}\,(m)^{k_1+\cdots+k_n}}{s(\Psi'_{\mathtt{N}}), \mathtt{x} :!^{k_1+1} s(!^{r_1}V_1) \vdash_{\mathtt{M}} \mathtt{N} : s(U_{\mathtt{N}})}\,(m)}{s(\Psi'_{\mathtt{N}}) \vdash_{\mathtt{M}} \lambda\mathtt{x}.\mathtt{N} :!^{k_1+1} s(!^{r_1}V_1) \multimap s(U_{\mathtt{N}})}\,(\multimap I)$$

and clearly $\mathfrak{A} \vdash_{\mathtt{M}} \lambda\mathtt{x}.\mathtt{N} : \sigma \multimap A$ can be obtained from it by applying repeatedly (m) and/or (w) rules.

Consider the case where Π ends as:

$$\dfrac{\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{N} : \sigma \multimap A \quad \mathfrak{B} \vdash_{\mathtt{M}} \mathtt{P} : \sigma \quad \mathfrak{A} \approx \mathfrak{B}}{\mathfrak{A}, \mathfrak{B} \vdash_{\mathtt{M}} \mathtt{NP} : A}\,(\multimap E)$$

Let $\mathbf{\Pi}(\mathtt{NP}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$, $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ and $\mathbf{\Pi}(\mathtt{P}) = \langle \Psi_{\mathtt{P}}, U_{\mathtt{P}}, \mathcal{G}_{\mathtt{P}}, \mathcal{F}_{\mathtt{P}}, \mathcal{H}_{\mathtt{P}} \rangle$. By definition of $\mathbf{\Pi}$ algorithm $\Psi = \Psi_{\mathtt{N}} \sqcup \Psi'_{\mathtt{P}}$ where, if $\Psi_{\mathtt{P}} = \mathtt{x}_1 : !^{p_1}V_1, \ldots, \mathtt{x}_n :!^{p_n}V_n$, then $\Psi'_{\mathtt{P}} = \mathtt{x}_1 :!^{q_1}V_1, \ldots, \mathtt{x}_n :!^{q_n}V_n$ for fresh $q_1, \ldots, q_n$. Moreover, for fresh u, t, a, b and p, if $\mathcal{I} = \mathtt{range}(\Psi)$ and $\mathtt{Unify}(\Psi_{\mathtt{N}}, \Psi'_{\mathtt{P}}, U_{\mathtt{N}}, !^{p}U_{\mathtt{P}} \multimap \forall t.a) = \mathcal{H}'$ then $U \equiv \forall u.b$, $\mathcal{G} = \mathcal{G}_{\mathtt{N}} \cup \mathcal{G}_{\mathtt{P}} \cup \{(\forall t.a, b)\}$, $\mathcal{F} = \mathcal{F}_{\mathtt{N}} + \mathcal{F}_{\mathtt{P}} + [u \mapsto \mathcal{I}]$ and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H}_{\mathtt{P}} \uplus \mathcal{H}' \uplus \{q_i \geq p_i + p\}$.

By induction hypothesis there exists a ground substitution $s_{\mathtt{N}}$ satisfying $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{N}}$ such that $\Sigma' \rhd s_{\mathtt{N}}(\Psi_{\mathtt{N}}) \vdash_{\mathtt{M}} \mathtt{N} : s_{\mathtt{N}}(U_{\mathtt{N}})$ and $\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{N} : \sigma \multimap A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. Analogously by induction hypothesis there exists a ground substitution $s_{\mathtt{P}}$ satisfying $\mathcal{G}_{\mathtt{P}}, \mathcal{F}_{\mathtt{P}}$ and $\mathcal{H}_{\mathtt{P}}$ such that: $\Theta' \rhd s_{\mathtt{P}}(S_{\mathtt{P}}(\Psi_{\mathtt{P}})) \vdash_{\mathtt{M}} \mathtt{P} : s_{\mathtt{P}}(S_{\mathtt{P}}(U_{\mathtt{P}}))$ and $\mathfrak{B} \vdash_{\mathtt{M}} \mathtt{P} : \sigma$ can be obtained from Θ' by applying repeatedly (w), (m) and/or (sp) rules.

Since $\mathbf{\Pi}(\mathtt{N})$ and $\mathbf{\Pi}(\mathtt{P})$ are distinct we can build a ground substitution s' acting as $s_{\mathtt{N}}$ on schemes in $\mathbf{\Pi}(\mathtt{N})$ and as $s_{\mathtt{P}}$ on schemes in $\mathbf{\Pi}(\mathtt{P})$. So s' in particular is such that $s'(\Psi_{\mathtt{N}}) \vdash_{\mathtt{M}} \mathtt{N} : s'(U_{\mathtt{N}})$ and $\mathfrak{A} \vdash_{\mathtt{M}} \mathtt{N} : \sigma \multimap A$ can be obtained from it by applying repeatedly (w) and/or (m) rules and moreover $s'(\Psi_{\mathtt{P}}) \vdash_{\mathtt{M}} \mathtt{P} : s'(U_{\mathtt{P}})$ and $\mathfrak{B} \vdash_{\mathtt{M}} \mathtt{P} : \sigma$ can be obtained from it by applying repeatedly (w), (m) and/or (sp) rules. Moreover, since t, a and p are fresh, we can choose s' satisfying also $s'(U_{\mathtt{N}}) \equiv \sigma \multimap A \equiv s'(!^{p}U_{\mathtt{P}} \multimap \forall t.a)$. Hence in particular s' satisfies $\mathcal{H}'$.

Since $u, b, q_1, \ldots, q_n$ are fresh, it is easy to extend s' to a ground substitution $s = s'[b \mapsto$

$s(\forall t.a), u \mapsto \epsilon, q_1 \mapsto s(p_1) + s(p), \ldots, q_n \mapsto s(p_n) + s(p)]$. Clearly s satisfies $\mathcal{G}, \mathcal{F}$ and $\mathcal{H}$. If $s(p) = k$, then the conclusion follows by Σ ending as:

$$\dfrac{\Sigma' \rhd s(\Psi_{\mathtt{N}}) \vdash_{\mathrm{M}} \mathtt{N} : s(U_{\mathtt{N}}) \qquad \dfrac{\Theta' \rhd s(\Psi_{\mathtt{P}}) \vdash_{\mathrm{M}} \mathtt{P} : s(U_{\mathtt{P}})}{!^k s(\Psi_{\mathtt{P}}) \vdash_{\mathrm{M}} \mathtt{P} : !^k s(U_{\mathtt{P}})}\,(sp)^k}{s(\Psi_{\mathtt{N}}), !^k s(\Psi_{\mathtt{P}}) \vdash_{\mathrm{M}} \mathtt{NP} : s(\forall t.a)}\,(\multimap E)$$

and $\mathfrak{A}, \mathfrak{B} \vdash_{\mathrm{M}} \mathtt{NP} : A$ can be obtained from it by applying repeatedly (w), (m) and/or (sp) rules.
Consider now the case where Π ends as:

$$\dfrac{\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{N} : \forall \alpha.A}{\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{N} : A[B/\alpha]}\,(\forall E)$$

We here need to proceed also by cases on the structure of $\mathtt{N}$.
Let $\mathtt{N} \equiv \mathtt{x}$. Then, $\mathbf{\Pi}(\mathtt{x}) = \langle \{\mathtt{x} :!^p \forall t.a\}, \forall u.b, \{(\forall t.a, b)\}, [u \mapsto \{\forall t.a\}], \{\emptyset, \emptyset, \{p \geq 0\}\}\rangle$. By induction hypothesis there exists a ground substitution s satisfying the constraints such that: $\Sigma' \rhd \mathtt{x} : s(!^p \forall t.a) \vdash_{\mathrm{M}} \mathtt{x} : s(\forall u.b)$ and $\mathfrak{A} \vdash_{\mathrm{M}} \mathtt{x} : \forall \alpha.A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules.
Clearly $A \equiv \forall \vec{\alpha}.C$ for some C non externally quantified. Then, since $s(\forall u.b) \equiv \forall \alpha.\forall \vec{\alpha}.C$ clearly $s = s''[u \mapsto \alpha, \vec{\alpha}, b \mapsto C]$ for some s''. Since u and b are fresh, we can build a substitution s' such that $s' = s''[u \mapsto \vec{\alpha}, b \mapsto C[B/\alpha]]$. Consider the scheme system $(\forall t.a, b)$, by hypothesis $s(\forall t.a) \leq C$, but s satisfies the constraints $[u \mapsto \{\forall t.a\}]$, so $\forall \beta \in s(u) : \beta \notin \mathrm{FTV}(s(\forall t.a))$ and hence in particular $\alpha \notin \mathrm{FTV}(s(\forall t.a))$. This implies that $s(\forall t.a) \leq C[B/\alpha]$. Then the conclusion follows by Σ ending as:

$$\dfrac{\Sigma' \rhd \mathtt{x} : s(!^p \forall t.a) \vdash_{\mathrm{M}} \mathtt{x} : s(\forall u.b) \equiv \forall \alpha.\vec{\alpha}.C}{\mathtt{x} : s(!^p \forall t.a) \vdash_{\mathrm{M}} \mathtt{x} : s'(\forall u.b) \equiv \forall \vec{\alpha}.C[B/\alpha]}\,(\forall E)$$

The case $\mathtt{N} \equiv \mathtt{PQ}$ is similar.
Consider the case $\mathtt{N} \equiv \lambda \mathtt{x}.\mathtt{P}$. Then, $\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{P}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H}\rangle$ where $U \equiv \forall u.\phi \multimap V$ for some ϕ, V and fresh u. By induction hypothesis there exists a ground substitution s satisfying the constraints such that: $\Sigma' \rhd s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : s(\forall u.\phi \multimap V)$ and $\mathfrak{A} \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : \forall \alpha.A$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. Hence in particular $s(\forall u.\phi \multimap V) \equiv \forall \alpha.A$. By Generation Lemma 40.1 $\Sigma' \rightsquigarrow \Sigma''$ ending as:

$$\dfrac{\Theta \rhd s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : A}{s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : \forall \alpha.A \equiv s(\forall u.\phi \multimap V)}\,\forall I$$

Moreover, since $\alpha \notin \mathrm{FTV}(s(\Psi))$ by Lemma 41 we have a derivation $\Theta \rhd s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : A[B/\alpha]$. Hence by induction hypothesis we have a ground substitution s' such

that there exists $\Theta \rhd s'(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : s'(\forall u.\phi \multimap V)$ and $s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : A[B/\alpha]$ can be obtained from Σ' by applying repeatedly (w), (m) and/or (sp) rules. Since also $\mathfrak{A} \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : A[B/\alpha]$ can be recovered from $s(\Psi) \vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{P} : A[B/\alpha]$ by applying repeatedly (w), (m) and/or (sp) rules, then the conclusion follows.
The case Π ends by $(\multimap I)$ rule is simpler. The cases where Π ends either by rule (w), (m) or (sp) follow easily by induction hypothesis. □

4.5. An example

Let we show by an example how the algorithm works. The example term is $\lambda \mathtt{x}.\mathtt{x}\mathtt{x}$. First of all $\mathbf{\Pi}(\mathtt{x}) = \langle \Psi_1, U_1, \mathcal{G}_1, \mathcal{F}_1, \mathcal{H}_1 \rangle$ where

$$\begin{aligned} \Psi_1 &= \{\mathtt{x} :!^{p_1} \forall t_1.a_1\} \\ U_1 &= \forall u_1.b_1 \\ \mathcal{G}_1 &= \{(\forall t_1.a_1, b_1)\} \\ \mathcal{F}_1 &= [u_1 \mapsto \{\forall t_1.a_1\}] \\ \mathcal{H}_1 &= \langle \emptyset, \emptyset, \{p_1 \geq 0\} \rangle \end{aligned}$$

and again $\mathbf{\Pi}(\mathtt{x}) = \langle \Psi_2, U_2, \mathcal{G}_2, \mathcal{F}_2, \mathcal{H}_2 \rangle$ where

$$\begin{aligned} \Psi_2 &= \{\mathtt{x} :!^{p_2} \forall t_2.a_2\} \\ U_2 &= \forall u_2.b_2 \\ \mathcal{G}_2 &= \{(\forall t_2.a_2, b_2)\} \\ \mathcal{F}_2 &= [u_2 \mapsto \{\forall t_2.a_2\}] \\ \mathcal{H}_2 &= \langle \emptyset, \emptyset, \{p_2 \geq 0\} \rangle \end{aligned}$$

Now $\mathtt{U}(\forall u_1.b_1, !^q(\forall u_2.b_2) \multimap \forall t.a) = \langle \mathcal{P}, \mathcal{Q}, \mathcal{C} \rangle$ where

$$\begin{aligned} \mathcal{P} &= \{b_1 =!^q (\forall u_2.b_2) \multimap \forall t.a\} \\ \mathcal{Q} &= \{u_1 = \varepsilon\} \\ \mathcal{C} &= \emptyset \end{aligned}$$

Moreover: $\mathtt{U}(!^{p_1} \forall t_1.a_1, !^{p_3} \forall t_2.a_2) = \langle \mathcal{P}', \mathcal{Q}', \mathcal{C}' \rangle$ where

$$\begin{aligned} \mathcal{P}' &= \{a_1 = a_2\} \\ \mathcal{Q}' &= \{t_1 = t_2\} \\ \mathcal{C}' &= \{p_1 = p_3\} \end{aligned}$$

hence we have

$$\mathtt{Unify}(\{\mathtt{x} :!^{p_1} \forall t_1.a_1\}, \{\mathtt{x} :!^{p_3} \forall t_2.a_2\}, \forall u_1.b_1, !^q(\forall u_2.b_2) \multimap \forall t.a) = \langle \mathcal{P} \cup \mathcal{P}', \mathcal{Q} \cup \mathcal{Q}', \mathcal{C} \cup \mathcal{C}' \rangle$$

So $\mathbf{\Pi}(\mathtt{xx}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$ where:

$$\begin{array}{rcl}
\Psi & = & \{\mathtt{x} :!^{p_1} \forall t_1.a_1, \mathtt{x} :!^{p_3} \forall t_2.a_2\} \\
U & = & \forall v.c \\
\mathcal{G} & = & \{(\forall t_1.a_1, b_1), (\forall t_2.a_2, b_2), (\forall t.a, c)\} \\
\mathcal{F} & = & \{u_1 \mapsto \{\forall t_1.a_1\}, u_2 \mapsto \{\forall t_2.a_2\}, v \mapsto \{\forall t_1.a_1, \forall t_2.a_2\}\} \\
\mathcal{H} & = & \langle \{b_1 =!^q (\forall u_2.b_2) \multimap \forall t.a, a_1 = a_2\}, \{u_1 = \varepsilon, t_1 = t_2\}, \\
 & & \{p_1 \geq 0, p_2 \geq 0, p_1 = p_3, p_3 \geq p_2 + q\} \rangle
\end{array}$$

concluding $\mathbf{\Pi}(\lambda \mathtt{x}.\mathtt{xx}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$ where:

$$\begin{array}{rcl}
\Psi & = & \emptyset \\
U & = & \forall w.!^r (\forall t_1.a_1) \multimap \forall v.c \\
\mathcal{G} & = & \{(\forall t_1.a_1, b_1), (\forall t_2.a_2, b_2), (\forall t.a, c)\} \\
\mathcal{F} & = & \{u_1 \mapsto \{\forall t_1.a_1\}, u_2 \mapsto \{\forall t_2.a_2\}, v \mapsto \{\forall t_1.a_1, \forall t_2.a_2\}\} \\
\mathcal{H} & = & \langle \{b_1 =!^q (\forall u_2.b_2) \multimap \forall t.a, a_1 = a_2\}, \{u_1 = \epsilon, t_1 = t_2\}, \\
 & & \{p_1 \geq 0, p_2 \geq 0, p_1 = p_3, p_3 \geq p_2 + q, r > p_1, r > p_3\} \rangle
\end{array}$$

The substitution $S = S_I[b_1 \mapsto !^q (\forall u_2.b_2) \multimap \forall t.a, a_1 \mapsto a_2, u_1 \mapsto \epsilon, t_1 \mapsto t_2, p_1 \mapsto p_3]$ clearly satisfies $\mathcal{H}$, and is such that:

$$\begin{array}{rcl}
S(U) & = & \forall w.!^r (\forall t_2.a_2) \multimap \forall v.c \\
S(\mathcal{G}) & = & \{(\forall t_2.a_2, !^q (\forall u_2.b_2) \multimap \forall t.a), (\forall t_2.a_2, b_2), (\forall t.a, c)\} \\
S(\mathcal{F}) & = & \{\epsilon \mapsto \{\forall t_2.a_2\}, u_2 \mapsto \{\forall t_2.a_2\}, v \mapsto \{\forall t_2.a_2\}\} \\
S(\mathcal{H}) & = & \langle \{!^q (\forall u_2.b_2) \multimap \forall t.a =!^q (\forall u_2.b_2) \multimap \forall t.a, a_2 = a_2\}, \{\epsilon = \epsilon, t_2 = t_2\}, \\
 & & \{p_3 \geq 0, p_2 \geq 0, p_3 = p_3, p_3 \geq p_2 + q, r > p_3\} \rangle
\end{array}$$

For every ground substitution s', the substitution $s = s'[a_2 \mapsto \alpha, b_2 \mapsto \beta, c \mapsto \gamma, a \mapsto \gamma, t_2 \mapsto \alpha, u_2 \mapsto \beta, v \mapsto \gamma, w \mapsto \epsilon, t \mapsto \epsilon, p_3 \mapsto 0, p_2 \mapsto 0, q \mapsto 0, r \mapsto 1]$ satisfies $S(\mathcal{G})$, $S(\mathcal{F})$ and $S(\mathcal{H})$. In fact, we have:

$$\begin{array}{rcl}
s(S(\mathcal{G})) & = & \{(\forall \alpha.\alpha, (\forall \beta.\beta) \multimap \gamma), (\forall \alpha.\alpha, \beta), (\gamma, \gamma)\} \\
s(S(\mathcal{F})) & = & \{\epsilon \mapsto \{\forall \alpha.\alpha\}, \beta \mapsto \{\forall \alpha.\alpha\}, \gamma \mapsto \{\forall \alpha.\alpha\}\} \\
s(S(\mathcal{H})) & = & \langle \{(\forall \beta.\beta) \multimap \gamma = (\forall \beta.\beta) \multimap \gamma, \alpha = \alpha\}, \{\epsilon = \epsilon, \alpha = \alpha\}, \\
 & & \{0 \geq 0, 0 \geq 0, 0 = 0, 0 \geq 0 + 0, 1 > 0\} \rangle
\end{array}$$

hence by Theorem 21 we have $\vdash_{\mathrm{M}} \lambda \mathtt{x}.\mathtt{xx} :!(\forall \alpha.\alpha) \multimap \forall \gamma.\gamma \equiv s(S(U))$, as we expected.

4.6. Deciding constraints

By Theorem 19, for every λ-term $\mathtt{M}$, the algorithm $\mathbf{\Pi}$ ends returning a set of constraints. Moreover, by the correctness result in Theorem 20 if there exists a scheme substitution

and a ground substitution satisfying these constraints then the term $\mathtt{M}$ is typable in $\mathrm{STA}_{\mathrm{M}}$. The correctness of the unification algorithm $\mathtt{U}$ expressed by Theorem 17 with Lemma 44 assures that a scheme substitution satisfying such kind of constraints always exists. Clearly the same cannot be concluded for the existence of a ground substitution satisfying the constraints.

In fact the type inference problem for $\mathrm{STA}_{\mathrm{M}}$ can be reformulated as the problem of deciding if a ground substitution satisfying the constraints exists. Deciding if a ground substitution satisfying a set of binding constraints and a constraints sequence is possible. On the other hand, deciding if a ground substitution satisfying a scheme system exists, seems problematic. In [Giannini and Ronchi Della Rocca, 1994] the authors have shown that the problem of satisfiability of generic scheme systems is equivalent to the semi-unification problem, hence undecidable. Since here the shape of scheme systems and the restrictions on types does not help, we conjecture that the satisfiability is still undecidable.

So following what has been done in [Giannini and Ronchi Della Rocca, 1994] for System F, for every $n \in \mathbb{N}$ we can define a bounded type containment relation $\leq_{\mathcal{J}}^{n}$ such that $\forall\vec{a}.A \leq_{\mathcal{J}}^{n} C$ if and only if $C \equiv A[\vec{B}/\vec{\alpha}]$ and the variables in $\vec{\alpha}$ occur in the syntax tree of A at a depth less or equal to n.

Then, we can define a countable set of type assignment systems $\mathrm{STA}_{\mathrm{M}}^{n}$ which is a complete stratification of the system $\mathrm{STA}_{\mathrm{M}}$. Roughly speaking, for each $n \in \mathbb{N}$, the system $\mathrm{STA}_{\mathrm{M}}^{n}$ is obtained by replacing the $(\forall E)$ rule in Table 3.2 by the following rule:

$$\frac{\Gamma \vdash_{\mathrm{M}} \mathtt{M} : A \quad A \leq_{\mathcal{J}}^{n} B}{\Gamma \vdash_{\mathrm{M}} \mathtt{M} : B}\ (n\text{-}\forall E)$$

hence every system $\mathrm{STA}_{\mathrm{M}}^{n}$ internalize the type containment $\leq_{\mathcal{J}}^{n}$.

We conjecture that in every $\mathrm{STA}_{\mathrm{M}}^{n}$ the type inference problem is decidable. Moreover the encoding of polynomial time Turing machines given in Chapter 2 for STA still works for every system $\mathrm{STA}_{\mathrm{M}}^{k}$ by taking $k \geq 11$. So, what we can gain is a system complete for polynomial time computation with decidable type inference. We leave this arguments for future investigations.

5. A Logical Account of PSPACE

5.1. Introduction

In this chapter we present the system $\mathrm{STA}_{\mathbf{B}}$: a type assignment system for a language extending λ-calculus by basic boolean constants that we prove correct and complete for polynomial space computations.
We start by introducing the syntax of the system and by giving some basic properties. Then, we define the operational semantics of $\mathrm{STA}_{\mathbf{B}}$ programs (i.e. closed terms typable with ground type) through a big step abstract machine. We show that every program can be executed in polynomial space in the size of the input, by the abstract machine. Moreover we show that $\mathrm{STA}_{\mathbf{B}}$ is also complete for PSPACE, by showing that all decision functions computable in polynomial space can be programmed. Finally some complementary argument are considered.

5.2. Soft Type Assignment system with Booleans

In this section we present the Soft Type Assignment system with Booleans ($\mathrm{STA}_{\mathbf{B}}$). $\mathrm{STA}_{\mathbf{B}}$ is an extension of $\mathrm{STA}_{\mathrm{N}}$, the natural deduction version of the type assignment system STA, introduced in Chapter 3. It is obtained from $\mathrm{STA}_{\mathrm{N}}$ by extending both the term language and the set of types. The term language of $\mathrm{STA}_{\mathbf{B}}$ is the λ-calculus extended by boolean constants $\mathtt{0}, 1$ and an `if` constructor, while types are defined analogously to $\mathrm{STA}_{\mathrm{N}}$ types considering also a constant type $\mathbf{B}$ for booleans.

5.2.1. The system $\mathrm{STA}_{\mathbf{B}}$

We extend the λ-calculus by basic boolean constants.

Definition 20. *Let $\mathcal{B} = \{\mathtt{0}, 1\}$ be the set of* booleans *and* $\mathtt{x}$ *range over a countable set of variables. Then, the set $\Lambda_{\mathcal{B}}$ of* $\mathrm{STA}_{\mathbf{B}}$-terms *is defined by the following grammar:*

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

Axiom and Weakening:

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

Multiplicatives:

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

Exponentials:

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

Quantifier:

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

Booleans:

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

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

Table 5.1.: The Soft Type Assignment system with Booleans

$|\mathtt{M}|$ *denotes the* size *of the term* $\mathtt{M}$ *and it is defined as:*

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

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

Terms are denoted by $\mathtt{M}, \mathtt{N}, \mathtt{V}, \mathtt{P}$. As usual the variable $\mathtt{x}$ in $\lambda\mathtt{x}.\mathtt{M}$ is said *bound* and 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 $\mathtt{M}$, while $n_o(\mathtt{x}, \mathtt{M})$ denotes the number of free occurrences of the variable $\mathtt{x}$ in $\mathtt{M}$.
We extend the set of STA types by a constant type for booleans.

Definition 21. *Let* $\mathbf{B}$ *be a* ground *type for booleans and* α *range over a countable set of type variables. Then,* linear $\mathrm{STA}_{\mathbf{B}}$-types *and* $\mathrm{STA}_{\mathbf{B}}$-types *are mutually inductively defined as:*

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

The set of $\mathrm{STA}_{\mathbf{B}}$-types *is denoted by* $\mathcal{T}_{\mathbf{B}}$.

Type variables are denoted by α, β, linear $\mathrm{STA}_{\mathbf{B}}$-types by A, B, C, and $\mathrm{STA}_{\mathbf{B}}$-types by σ, τ, μ. In the sequel we usually omit the prefix $\mathrm{STA}_{\mathbf{B}}$ and we simply write linear types and types respectively. As usual the variable α in the type $\forall\alpha.A$ is said *bound* and types will be considered modulo renaming of bound variables. $\sigma[A/\alpha]$ denotes the capture free substitution in σ of all occurrences of the type variable α by the linear type A. $\mathrm{FTV}(\sigma)$ denotes the set of free type variables of σ. As usual $\multimap$ associates to the right and has precedence on $\forall$, while ! has precedence on everything else. In what follows $\forall\vec{\alpha}.A$ is an abbreviation for $\forall\alpha_1....\forall\alpha_m.A$, and $!^n\sigma$ is an abbreviation for $!...!\sigma$ n-times. Note that each type has the shape $!^n\forall\vec{\alpha}.A$.
The symbol $\equiv$ denotes the syntactical equality both for types and terms, modulo renaming of bound variables.
A context Γ is a set of type assignment assumptions of the shape $\mathtt{x} : \sigma$, where all variables are different. By abuse of notation contexts are denoted by Γ, Δ. $\mathrm{dom}(\Gamma)$ and $\mathrm{FTV}(\Gamma)$ denote respectively the set $\{\mathtt{x} \mid \exists \mathtt{x} : \sigma \in \Gamma\}$ and the set $\{\alpha \in \mathrm{FTV}(\sigma) \mid \exists \mathtt{x} : \sigma \in \Gamma\}$. We write $\Gamma\#\Delta$ if $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$.
$\mathrm{STA}_{\mathbf{B}}$ proves sequents of the shape:

$$\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$$

where Γ is a context, $\mathtt{M}$ is a term, and σ is a $\mathrm{STA}_{\mathbf{B}}$-type. The rules defining $\mathrm{STA}_{\mathbf{B}}$ are depicted in Table 5.1.
Derivations in $\mathrm{STA}_{\mathbf{B}}$ are denoted by Π, Σ, Θ. $\Pi \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$ denotes a derivation Π with conclusion $\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$. As usual $\vdash_{\mathrm{B}} \mathtt{M} : \sigma$ is a short for $\emptyset \vdash_{\mathrm{B}} \mathtt{M} : \sigma$.
It is worth noting that while all rules in STA have a multiplicative treatment of contexts, the rule $(\mathbf{B}E)$ of $\mathrm{STA}_{\mathbf{B}}$ uses an additive treatment of contexts and so contraction is free. Clearly $\mathrm{STA}_{\mathrm{N}}$ is a subsystem of $\mathrm{STA}_{\mathbf{B}}$. So we have the following obvious result.

Lemma 46. *$\Gamma \vdash_{\mathrm{T}} \mathtt{M} : \sigma$ implies $\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$*

Proof. Obvious, by Theorem 10. □

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

Lemma 47 (Free variables lemma).

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

Proof. All the three points can be easily proved by induction on the derivation proving $\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$. □

Moreover, we have the following analogous of Lemma 5.

Lemma 48. *$\Gamma, \mathtt{x} : A \vdash_{\mathrm{B}} \mathtt{M} :!\sigma$ implies $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.*

Proof. Easy, by induction on the derivation proving $\Gamma, \mathtt{x} : A \vdash_{\mathrm{B}} \mathtt{M} :!\sigma$. □

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

Definition 22. *The reduction relation $\to_{\beta\delta} \subseteq \Lambda_{\mathcal{B}} \times \Lambda_{\mathcal{B}}$ is the contextual closure of the following rules:*

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

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

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

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

Analogously to the case of STA, in what follows, we will need to talk about proofs modulo commutations of rules.

Definition 23. *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.*

In the sequel we need an analogous of the Generation Lemma 8 in order to connect the shape of a term with its possible typings.

Lemma 49 (Generation lemma).

1. $\Pi \rhd \Gamma \vdash_{\mathrm{B}} \lambda \mathtt{y}.\mathtt{M} : \forall\alpha.A$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$ *where the last rule of* Π' *is* $(\forall I)$*.*

2. $\Pi \rhd \Gamma \vdash_{\mathrm{B}} \lambda \mathtt{y}.\mathtt{M} : \sigma \multimap A$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* $(\multimap I)$*.*

3. $\Pi \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M} : !\sigma$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$ *where* Π' *consists of a subderivation, ending with the rule* (sp) *proving* $!\Gamma' \vdash_{\mathrm{B}} \mathtt{M} : !\sigma$*, followed by a (maybe empty) sequence* δ *of rules* (w) *and/or* (m)*, dealing with variables not occurring in* M*.*

4. $\Pi \rhd !\Gamma \vdash_{\mathrm{B}} \mathtt{M} : !\sigma$ *implies there is* Π' *such that* $\Pi \rightsquigarrow \Pi'$*, whose last rule is* (sp)*.*

Proof.

1. Analogous to the proof of Lemma 8.6. By induction on Π. If the last applied rule 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:

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

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

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

   Then, the desired Π' is:

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

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

2. Analogous to the proof of Lemma 8.2 and similar to the proof of the previous point of this lemma.

3. Analogous to the proof of Lemma 8.4. By induction on Π. In the case the last applied rule is (sp), the proof is obvious. The case where the last applied rule 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 applied rule is:

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

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 ending with a rule (sp) proving $!\Delta_1 \vdash_{\mathrm{T}} \mathtt{N} :!\sigma$, 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 $\mathtt{x}_i \in \mathrm{FV}(\mathtt{N})$ then necessary $\mathtt{x}_i : \tau' \in \Delta_1$ and $\tau = !\tau'$. Let Δ_2 be the context $\Delta_1 - \{\mathtt{x}_1 : \tau_1, \ldots, \mathtt{x}_n : \tau_n\}$, then the conclusion follows by the derivation:

$$\dfrac{\dfrac{\Delta_2, \mathtt{x}_1 : \tau', \ldots, \mathtt{x}_n : \tau' \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{\Delta_2, \mathtt{x} :!\tau' \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] : \sigma}\ (m)}{!\Delta_2, \mathtt{x} :!\tau \vdash_{\mathrm{T}} \mathtt{N}[\mathtt{x}/\mathtt{x}_1, \cdots, \mathtt{x}/\mathtt{x}_n] :!\sigma}\ (sp)$$

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

4. Analogous to the proof of Lemma 8.5. By induction on Π. In the case the last applied rule is (sp), the proof is obvious. The only other possible case is when the last applied rule is (m). Consider the case where $\mathtt{M} \equiv \mathtt{N}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}/\mathtt{x}_n]$ and the last applied rule is:

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

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

$$\frac{\Theta \rhd \Delta, \mathtt{x}_1 : \tau', ..., \mathtt{x}_n : \tau' \vdash_{\mathrm{T}} \mathtt{N} : \sigma}{!\Delta, \mathtt{x}_1 :!\tau, ..., \mathtt{x}_n :!\tau \vdash_{\mathrm{T}} \mathtt{N} :!\sigma}\ (sp)$$

So the desired derivation Π' is Θ, followed by a rule (m) and a rule (sp). In the case τ is linear, by Lemma 48, $\mathtt{x}_i \notin \mathrm{FV}(\mathtt{M}')$ for each $1 \leq i \leq n$. Moreover by the previous point of this lemma, Σ can be rewritten as:

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

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).

□

5.2.2. Some properties

We would now prove that $\mathrm{STA}_{\mathbf{B}}$ enjoys the subject reduction property. The proof is similar to the one for STA. In particular the substitution lemma will be the key lemma. Instead of introducing a notion of chain analogous to the one defined in Section 2.3.1 of STA, here we can restrict our attention to the simpler technical notion of height of a variable in a derivation. This thanks to the fact that $\mathrm{STA}_{\mathbf{B}}$, extending $\mathrm{STA}_{\mathrm{N}}$, is a natural deduction system.

Definition 24. *Let* $\Pi \rhd \Gamma, \mathtt{x} : \tau \vdash_{\mathrm{B}} \mathtt{M} : \sigma$. *The* height *of* $\mathtt{x}$ *in* Π *is inductively defined as follows:*

- *if the last applied rule of* Π *is:*

$$\frac{}{\mathtt{x} : A \vdash_{\mathrm{B}} \mathtt{x} : A} \quad \textit{or} \quad \frac{\Gamma' \vdash_{\mathrm{B}} \mathtt{N} : \sigma}{\Gamma', \mathtt{x} : A \vdash_{\mathrm{B}} \mathtt{N} : \sigma}$$

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

- *if the last applied rule of* Π *is:*

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

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

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

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M} : \mathbf{B} \quad \Theta_0 \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_0 : A \quad \Theta_1 \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_1 : A}{\Gamma, \mathtt{x} : \tau \vdash_{\mathrm{B}} \ \mathtt{if\ M\ then\ N}_0\ \mathtt{else\ N}_1\ : A}$$

  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 an assumption with subject* $\mathtt{x}$ *both in the conclusion of the rule and in 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 50 (Substitution lemma). *Let* $\Pi \rhd \Gamma, \mathtt{x} : \mu \vdash_{\mathrm{B}} \mathtt{M} : \sigma$ *and* $\Sigma \rhd \Delta \vdash_{\mathrm{B}} \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$*. Then there exists* Θ *proving:*

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

Proof. 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 $\Pi' \rhd \Gamma', \mathtt{x} : \mu' \vdash_{\mathrm{B}} \mathtt{M} : \sigma'$ then by Lemma 49.3 $\Sigma \rightsquigarrow \Sigma''$ which is composed by a subderivation ending with an (sp) rule with premise $\Sigma' \rhd \Delta' \vdash_{\mathrm{B}} \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). By the induction hypothesis we have a derivation $\Theta' \rhd \Gamma', \Delta' \vdash_{\mathrm{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma'$. By applying the rule (sp) and the sequence of (w) and/or (m) rules we obtain $\Theta \rhd \Gamma, \Delta \vdash_{\mathrm{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$.
Consider the case Π ends by:

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

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

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

Consider the case Π ends by:

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

By Lemma 49.3 $\Sigma \rightsquigarrow \Sigma''$ ending by an (sp) rule with premise $\Sigma' \rhd \Delta' \vdash_{\mathrm{B}} \mathtt{N} : \mu'$ followed by a sequence of rules (w) and/or (m). Consider fresh copies of the derivation Σ' i.e. $\Sigma'_j \rhd \Delta'_j \vdash_{\mathrm{B}} \mathtt{N}_j : \mu'$ 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:

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

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

We can finally prove the main property of this section.

Lemma 51 (Subject Reduction). *Let* $\Gamma \vdash_{\mathtt{B}} \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$. *Then* $\Gamma \vdash_{\mathtt{B}} \mathtt{N} : \sigma$.

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

$$\frac{\Pi \rhd \Gamma \vdash_{\mathtt{B}} \mathtt{b} : \mathbf{B} \quad \Pi_{\mathtt{0}} \rhd \Gamma \vdash_{\mathtt{B}} \mathtt{M}_{\mathtt{0}} : A \quad \Pi_{\mathtt{1}} \rhd \Gamma \vdash_{\mathtt{B}} \mathtt{M}_{\mathtt{1}} : A}{\Gamma \vdash_{\mathtt{B}} \ \mathtt{if\ b\ then\ M_0\ else\ M_1}\ : A}\ (\mathbf{B}E)$$

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

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

where $\Gamma = \Gamma_1, \Gamma_2$. The others follow directly by induction hypothesis. Clearly $(\lambda\mathtt{x}.\mathtt{M})\mathtt{N} \to_\beta \mathtt{M}[\mathtt{N}/\mathtt{x}]$. By Lemma 49.2 $\Pi \rightsquigarrow \Pi_1$ ending as

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

By the Substitution Lemma 50 since $\Pi_2 \rhd \Gamma_1, \mathtt{x} : \sigma \vdash_{\mathtt{B}} \mathtt{M} : A$ and $\Sigma \rhd \Gamma_2 \vdash_{\mathtt{B}} \mathtt{N} : \sigma$ we have $\Gamma_1, \Gamma_2 \vdash_{\mathtt{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : A$, hence the conclusion follows. □

By strong normalization of STA we have the following.

Lemma 52 (Strong Normalization). *Let* $\Gamma \vdash_{\mathtt{B}} \mathtt{M} : \sigma$ *then* $\mathtt{M}$ *is strongly normalizing with respect to the reduction relation* $\to_{\beta\delta}$.

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 by derivations with a priori fixed degree, i.e. the nesting of (sp) applications.

Example 6. *Consider for* $n \in \mathbb{N}$ *terms* $\mathtt{M}_n$ *of the shape:*

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

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

5.3. Structural Operational Semantics

In this section the operational semantics of $\mathrm{STA}_{\mathbf{B}}$ programs is given, through an evaluation machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$, defined in SOS style [Plotkin, 2004, Kahn, 1987], performing the evaluation according to the leftmost outermost strategy. The machine, if restricted to λ-calculus, 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 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 typable with ground type) can be evaluated.

5.3.1. Memory devices

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

Definition 25.

- *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* $\mathrm{Ctx}_{\mathrm{m}}$. *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* $\mathbf{B}$-contexts *is defined by the following grammar:*

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

  The size of a $\mathbf{B}$*-context* $\mathcal{C}[\circ]$ *denoted* $|\mathcal{C}[\circ]|$ *is the size of the term obtained by replacing the symbol* $\circ$ *by a variable.*
  The cardinality of a $\mathbf{B}$*-context* $\mathcal{C}[\circ]$*, denoted by* $\#(\mathcal{C}[\circ])$*, is the number of nested* $\mathbf{B}$*-contexts in it. i.e.* $\#(\circ) = 1$, $\#((\ \mathtt{if}\ \mathcal{C}[\circ]\ \mathtt{then}\ \mathtt{M}\ \mathtt{else}\ \mathtt{N}\ )\mathtt{V}_1 \cdots \mathtt{V}_n) = \#(\mathcal{C}[\circ]) + 1$

ε denotes the empty m-context and $\mathcal{A}_1 @ \mathcal{A}_2$ denotes the concatenation of the m-contexts $\mathcal{A}_1$ and $\mathcal{A}_2$. Moreover, $[\mathtt{x} := \mathtt{M}] \in \mathcal{A}$ denotes the fact that $\mathtt{x} := \mathtt{M}$ is in the m-context $\mathcal{A}$. $\mathrm{FV}(\mathcal{A})$ denotes the set: $\bigcup_{[\mathtt{x}:=\mathtt{M}]\in\mathcal{A}} \mathrm{FV}(\mathtt{M})$.
As usual $\mathcal{C}[\mathtt{M}]$ denotes the term obtained by filling the hole $[\circ]$ in $\mathcal{C}[\circ]$ by $\mathtt{M}$. In general we

omit the hole $[\circ]$ and we range over $\mathbf{B}$-contexts by $\mathcal{C}$. $\mathtt{FV}(\mathcal{C})$ denotes the set $\mathtt{FV}(\mathcal{C}[\mathtt{M}])$ 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 26. *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 $\mathtt{M}[\mathtt{N}_n/\mathtt{x}_n][\mathtt{N}_{n-1}/\mathtt{x}_{n-1}]\cdots[\mathtt{N}_1/\mathtt{x}_1]$.*

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

In defining the operational semantics of $\mathrm{STA}_{\mathbf{B}}$ we restrict our attention to programs.

Definition 27. *The set $\mathcal{P}$ of* $\mathrm{STA}_{\mathbf{B}}$ programs *is the set of closed terms typable by the ground type. i.e. $\mathcal{P} = \{\mathtt{M} \mid \vdash_{\mathrm{B}} \mathtt{M} : \mathbf{B}\}$.*

Moreover the following characterization of terms underlies the design of the evaluation machine.

Lemma 53. *Let $\mathtt{M} \in \Lambda_{\mathcal{B}}$ such that $\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$. Then, $\mathtt{M}$ has the following shape:*

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

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 $\mathtt{if\ P\ then\ N_0\ else\ N_1}$. Moreover if $\mathtt{M} \in \mathcal{P}$ then $n = 0$, while if ζ is a boolean $\mathtt{b}$ then $m = 0$.

Proof. By induction on $\mathtt{M}$ using the grammar of Definition 20. □

The rules defining $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ are depicted in Table 5.2. They need some comments. 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 to it in the m-context. Rules $(\mathtt{if}\ \mathtt{0})$ and $(\mathtt{if}\ 1)$ perform the δ reductions. Here the evaluation naturally erases part of the subject, but the erased information is stored in the $\mathbf{B}$-context.

In order to state formally the behaviour of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ we need the following further definition.

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

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

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

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

$$\frac{\mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m], \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{1} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\mathtt{if}\ \mathtt{1})$$

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

Table 5.2.: The Abstract Machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$

Definition 28.

- *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* $\mathtt{M} \Downarrow$*. As usual* $\models \mathtt{M} \Downarrow b$ *is a short for* $[\circ], \varepsilon \models \mathtt{M} \Downarrow b$*.*

- *Derivation trees in the abstract machine are called* computations *and are denoted by* $\nabla, \diamond$*.* $\nabla :: \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *denotes a computation with conclusion* $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$*.*

- *Given a computation* ∇ *each node of* ∇*, which is of the shape* $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *is a* configuration*.* $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b} \in \nabla$ *denotes that* $\mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *is a configuration in the computation* ∇*. Configurations are ranged over by* ϕ, ψ*.* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ *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*.*

- *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 write* $\mathtt{path}(\phi)$ *when* ∇ *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 5.2, and from the previous comments, it is easy to verify the following.

Lemma 54.

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

Proof.

1. By induction on the height of the subderivation $\diamond$ of ∇ proving $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}'$. The base case is trivial. Let $\diamond$ ends as:
$$\frac{\mathcal{C}, \mathcal{A}@[\mathtt{x}' := \mathtt{N}] \models \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}{\phi \succ \mathcal{C}, \mathcal{A} \models (\lambda\mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}\ (\beta)$$
By induction hypothesis $(\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{N}]} \to^*_{\beta\delta} \mathtt{b}'$. Clearly since $\mathtt{x}'$ is fresh:
$$(\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@[\mathtt{x}':=\mathtt{N}]} \equiv ((\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)[\mathtt{N}/\mathtt{x}'])^{\mathcal{A}} \equiv (\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$
hence:
$$((\lambda\mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to_{\beta\delta} (\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$$
and the conclusion follows.
The case of a rule (h) follows directly by induction hypothesis.
Let $\diamond$ ends as:
$$\frac{\mathcal{C}', \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}{\phi \succ \mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}\ (\mathtt{if}\ \mathtt{0})$$
where $\mathcal{C}' = \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$. By induction hypothesis $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{0}$, hence:
$$((\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} ((\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$
and by δ reduction
$$((\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to_{\delta} (\mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}}$$
moreover, since by induction hypothesis we also have $(\mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}'$, the conclusion follows. The case $\diamond$ ends by $(\mathtt{if}\ \mathtt{1})$ rule is analogous.

2. By induction on the height of ∇. The base case is trivial, so consider the case the height of ∇ is greater than 1. We proceed by induction on $\mathtt{path}(\phi)$. The case $\mathtt{path}(\phi) = \mathtt{0}$ is trivial, so suppose $\mathtt{path}(\phi) = \mathtt{n} + 1$.
Consider we are in the case:
$$\frac{\phi \succ \mathcal{C}, \mathcal{A}@[\mathtt{x}' := \mathtt{N}] \models \mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}{\psi \succ \mathcal{C}, \mathcal{A} \models (\lambda\mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}'}\ (\beta)$$
By induction hypothesis since $\mathtt{path}(\psi) = \mathtt{n}$, we have:
$$\mathtt{M} \to^*_{\beta\delta} (\mathcal{C}[(\lambda\mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$$
Clearly since $\mathtt{x}'$ is fresh, we have:
$$(\mathcal{C}[\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}@[\mathtt{x}':=\mathtt{N}]} \equiv ((\mathcal{C}[\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m])[\mathtt{N}/\mathtt{x}'])^{\mathcal{A}} \equiv (\mathcal{C}[\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}}$$
hence by β-reduction and confluence:
$$\mathtt{M} \to^*_{\beta\delta} (\mathcal{C}[(\lambda\mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \to_{\beta} (\mathcal{C}[\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$$
and the conclusion follows.
The case of a rule (h) is simpler since by definition, the variables in $\mathcal{A}$ are all distinct.
The case:
$$\frac{\phi \succ \mathcal{C}', \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\psi \succ \mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\mathtt{if}\ \mathtt{0})$$
where $\mathcal{C}' = \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$ follows directly by induction hypothesis since clearly $(\mathcal{C}[(\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \equiv (\mathcal{C}'[\mathtt{M}])^{\mathcal{A}}$.
Now, consider we are in the case:
$$\frac{\mathcal{C}', \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0} \quad \phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}{\psi \succ \mathcal{C}, \mathcal{A} \models (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \Downarrow \mathtt{b}}\ (\mathtt{if}\ \mathtt{0})$$
where $\mathcal{C}' = \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m]$. By induction hypothesis since $\mathtt{path}(\psi) = \mathtt{n}$:
$$\mathtt{M} \to^*_{\beta\delta} (\mathcal{C}[(\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b}$$
moreover, by the previous point of this lemma $(\mathtt{M})^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{0}$. So by δ-reduction and confluence:
$$\begin{array}{rl} \mathtt{M} \to^*_{\beta\delta} & (\mathcal{C}[(\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \\ \to^*_{\beta\delta} & (\mathcal{C}[(\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \\ \to_{\delta} & (\mathcal{C}[\mathtt{N}_{\mathtt{0}}\mathtt{V}_1 \cdots \mathtt{V}_m])^{\mathcal{A}} \to^*_{\beta\delta} \mathtt{b} \end{array}$$
and the conclusion follows.

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

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

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

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

Table 5.3.: An example of computation in $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$.

3. By the previous points of this lemma and Lemma 51. □

Note that, in the previous lemma, the m-context and the **B**-context are essential in proving the desired properties. In fact the **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 7. *In Table 5.3 we present an example of* $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$ *computation on the same term of Example 6.*

Note that by Definition 28.1 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 22.

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

Proof. The proof of the theorem proceed by mixing induction on the shape of $\mathtt{M}$ and induction on the number of reduction steps to normal form, since by Lemma 52, $\mathtt{M}$ is strongly normalizing. Considering the characterization of Lemma 53, since $\mathtt{M} \in \mathcal{P}$, then it has the shape $\zeta \mathtt{V}_1 \cdots \mathtt{V}_n$ where ζ is either a boolean or a redex $(\lambda \mathtt{x}.\mathtt{N})\mathtt{P}$ or a term of the shape $\mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1$.
The base case is trivial by (Ax) rule. So consider the case $\mathtt{M} \equiv (\lambda \mathtt{x}.\mathtt{N})\mathtt{P}\mathtt{V}_1 \cdots \mathtt{V}_n$. Clearly $(\lambda \mathtt{x}.\mathtt{N})\mathtt{P}\mathtt{V}_1 \cdots \mathtt{V}_n \to_{\beta\delta} \mathtt{N}[\mathtt{P}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_n$ hence by induction hypothesis there exists $\mathtt{b}$ such that $\models \mathtt{N}[\mathtt{P}/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_n \Downarrow \mathtt{b}$. Now, for a fresh $\mathtt{x}'$, clearly:

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

so, in particular $\circ, [\mathtt{x}' := \mathtt{P}] \models \mathtt{N}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_n \Downarrow \mathtt{b}$ and by applying rule (β) the conclusion follows.
Now let $\mathtt{M} \equiv (\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n$. Clearly $\mathtt{M}' \in \mathcal{P}$, hence by induction hypothesis $\models \mathtt{M}' \Downarrow \mathtt{b}$ for some boolean $\mathtt{b}$, and by Lemma 54.1 $\mathtt{M}' \to^*_{\beta\delta} \mathtt{b}$. Then, in particular $(\ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n \to^*_{\beta\delta} (\ \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n$ and by δ-reduction $(\ \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n \to_\delta \mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n$ hence, by induction hypothesis $\mathtt{N}_{\mathtt{b}}\mathtt{V}_1 \cdots \mathtt{V}_n \Downarrow \mathtt{b}'$ for some $\mathtt{b}'$. So by rule $(\mathtt{if}\ \mathtt{b})$ the conclusion follows. □

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

In Table 5.4 we give a small step version of the machine $\mathrm{K}^{\mathcal{C}}_{\mathcal{B}}$. The rules are similar to the rules in Table 5.2 but the use of a garbage collector procedure described in Table 5.5 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.
We have introduced the small step machine since it explicits the evaluation order clarifying 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.

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

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

$$\frac{\mathcal{C}' \equiv \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n]}{\langle \mathcal{C}, \mathcal{A} \succ (\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_m \rangle \mapsto \langle \mathcal{C}', \mathcal{A} \succ \mathtt{M} \rangle} \ (\mathtt{if})$$

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

$$\frac{\mathcal{A}' = \mathtt{clear}(\mathcal{C}, \mathcal{A}, \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_n)}{\langle \mathcal{C}[(\ \mathtt{if}\ [\circ]\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )\mathtt{V}_1 \cdots \mathtt{V}_n], \mathcal{A} \succ 1 \rangle \mapsto \langle \mathcal{C}, \mathcal{A}' \succ \mathtt{N}_1\mathtt{V}_1 \cdots \mathtt{V}_n \rangle} \ (r_1)$$

Table 5.4.: The small step machine $\mathrm{k}^{\mathcal{C}}_{\mathcal{B}}$

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

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

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

Table 5.5.: The garbage collector procedure.

5.3.4. Space Measures

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

Definition 29. *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 $\nabla :: \mathcal{C}, \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{b}$ be a computation, then its* space occupation *denoted $\mathtt{space}(\nabla)$ is the maximal size of a configuration in ∇.*

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

Example 8. *By returning to the computation of Example 7 it is worth noting that to pass from the configuration ϕ to the configuration ψ (as in Table 5.3) all necessary informations are already present in the configuration ϕ itself. We can view such a step as a $\rightarrow_\delta$ step $(\ \mathtt{if}\ \mathtt{0}\ \mathtt{then}\ \mathtt{x}_1\ \mathtt{else}\ \mathtt{x}_1\ )^{\mathcal{A}_3} \rightarrow_\delta (\mathtt{x}_1)^{\mathcal{A}_3}$ noting that $(\mathtt{x}_1)^{\mathcal{A}_3} \equiv (\mathtt{x}_1)^{\mathcal{A}_2}$. In fact 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 introduce some relations between the size of the contexts and the behaviour of the machine, which will be useful later.

Definition 30. *Let ∇ be a computation and $\phi \in \nabla$ 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 55. *Let $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$. Then for each configuration $\phi \rhd \mathcal{C}_i, \mathcal{A}_i \models \mathtt{P}_i \Downarrow \mathtt{b}' \in \nabla$:*

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 $\mathbf{B}$-contexts can grow only by applications of $(\mathtt{if}\ \mathtt{0})$ and $(\mathtt{if}\ \mathtt{1})$ rules. □

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

Property 1. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$ *then for each* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$ *if* $[x_j := \mathtt{N}_j] \in \mathcal{A}$ *then* $\mathtt{N}_j$ *is an instance 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 ∇, 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 makes 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:

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

with $k \geq 1$ and $\mathtt{N}_j$ is an instance of 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 $\mathbf{B}$-context and of the subject of a configuration.

Lemma 56. *Let* $\mathtt{M} \in \mathcal{P}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$ *then for each configuration* $\phi \rhd \mathcal{C}, \mathcal{A} \models \mathtt{P} \Downarrow \mathtt{b}' \in \nabla$*:*

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

Proof.

1. By Lemma 55.1 and Property 1.
2. By inspection of the rules of Table 5.2 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. The conclusion follows directly by Lemma 55.1. □

5.4. PSPACE Soundness

In this section we 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. In fact, analogously to STA, data types are typable through derivations with degree 0. Hence the degree can be considered as a real characteristic of the program code.

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

Consider the following measure on $\mathrm{STA}_{\mathbf{B}}$ derivations.

Definition 31. *The degree* $\mathrm{d}(\Pi)$ *of a* $\mathrm{STA}_{\mathbf{B}}$ *derivation* Π *is the maximum nesting of applications of rule* (sp) *in* Π.

It is easy to verify that 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 32. *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.

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

The next lemma follows easily by the above definition.

Lemma 57. $\mathcal{P} = \bigcup_{n \in \mathbb{N}} \mathcal{P}_n$.

We here need to define measures of both terms and proofs, which are an adaptation of those given in Definition 8.

Definition 33.

- *The* rank *of a rule* (m) *as:*

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

  is the number $k \leq n$ *of variables* $\mathtt{x}_i$ *such that* $\mathtt{x}_i \in \mathrm{FV}(\mathtt{M})$ *for* $1 \leq i \leq n$.
  Let r *be the the maximum rank of a rule* (m) *in* Π. *The rank* $\mathrm{rk}(\Pi)$ *of* Π *is the maximum between* 1 *and* r.

- *Let* r *be a natural number. The* space weight $\delta(\Pi, r)$ *of* Π *with respect to* r *is defined inductively as follows.*
  - *If the last applied rule is* $(Ax), (\mathbf{B}_{\mathtt{0}} I), (\mathbf{B}_{\mathtt{1}} I)$ *then* $\delta(\Pi, r) = 1$.
  - *If the last applied rule is:*

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

    then $\delta(\Pi, r) = \delta(\Sigma, r) + 1$.
  - *If the last applied rule is:*

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

    then $\delta(\Pi, r) = r\delta(\Sigma, r)$.
  - *If the last applied rule is:*

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

    then $\delta(\Pi, r) = \delta(\Sigma, r) + \delta(\Theta, r) + 1$.
  - *If the last applied rule is:*

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M} : \mathbf{B} \quad \Theta_{\mathtt{0}} \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_{\mathtt{0}} : A \quad \Theta_{\mathtt{1}} \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_{\mathtt{1}} : A}{\Gamma \vdash_{\mathrm{B}} \ \mathtt{if\ M\ then\ N_0\ else\ N_1} \ : A}$$

    then $\delta(\Pi, r) = \max\{\delta(\Sigma, r), \delta(\Theta_{\mathtt{0}}, r), \delta(\Theta_{\mathtt{1}}, r)\} + 1$
  - *In every other case* $\delta(\Pi, r) = \delta(\Sigma, r)$ *where* Σ *is the unique premise derivation.*

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

Lemma 58 (Weighted Substitution Lemma). *Let* $\Pi \rhd \Gamma, \mathtt{x} : \mu \vdash_{\mathrm{B}} \mathtt{M} : \sigma$ *and* $\Sigma \rhd \Delta \vdash_{\mathrm{B}} \mathtt{N} : \mu$ *such that* $\Gamma \# \Delta$ *then there exists* $\Theta \rhd \Gamma, \Delta \vdash_{\mathrm{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : \sigma$ *such that for each* $r \geq \mathtt{rk}(\Pi)$*:*

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

Proof. It suffices to verify how the weights are modified by the proof of Lemma 50. We will use exactly the same notation as in the lemma.
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.
If Π ends by (sp) rule: $\delta(\Pi, r) = r\delta(\Pi', r)$ 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):

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

If Π ends by $(\mathbf{B}E)$: $\delta(\Pi, r) = \max_{0 \leq i \leq 2}(\delta(\Pi_i, r)) + 1$. By induction hypothesis we have derivations $\delta(\Theta_i, r) \leq \delta(\Pi_i, r) + \delta(\Sigma, r)$ for $0 \leq i \leq 2$. and applying a $(\mathbf{B}E)$ rule:

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

If Π ends by (m): $\delta(\Pi, r) = \delta(\Pi', r)$ and $\delta(\Sigma, r) = r\delta(\Sigma', r)$. Clearly $\delta(\Sigma', r) = \delta(\Sigma'_j, r)$ so $\delta(\Theta', r) \leq \delta(\Pi', r) + m\delta(\Sigma', r)$ and since $r \geq \mathtt{rk}(\Pi)$ then:

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

Now the rules (m) and (w) leave the space weight δ unchanged hence 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 59. *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 derivations $\Sigma \rhd\vdash_{\mathrm{B}} ((\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}} : \mathbf{B}$ *there exists a derivation* $\Theta \rhd\vdash_{\mathrm{B}} (\mathtt{M}[\mathtt{x}'/\mathtt{x}]\mathtt{V}_1 \cdots \mathtt{V}_m)^{\mathcal{A}@\{\mathtt{x}' := \mathtt{N}\}} : \mathbf{B}$ *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:*

$$\frac{\mathcal{C}', \mathcal{A} \models \mathtt{M} \Downarrow \mathtt{0} \quad \mathcal{C}, \mathcal{A} \models \mathtt{N_0 V_1 \cdots V_m} \Downarrow \mathtt{b}}{\mathcal{C}, \mathcal{A} \models (\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ )\mathtt{V_1 \cdots V_m} \Downarrow \mathtt{b}}$$

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* $\Sigma \rhd \vdash_{\mathtt{B}} ((\ \mathtt{if\ M\ then\ N_0\ else\ N_1}\ )\mathtt{V_1 \cdots V_m})^{\mathcal{A}} : \mathbf{B}$ *there are derivations* $\Theta \rhd \vdash_{\mathtt{B}} (\mathtt{M})^{\mathcal{A}} : \mathbf{B}$ *and* $\Pi \rhd \vdash_{\mathtt{B}} (\mathtt{N_0 V_1 \cdots V_m})^{\mathcal{A}} : \mathbf{B}$ *such that for every* $r \geq \mathbf{rk}(\Sigma)$*:*

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

Proof.

1. We can consider the case where $m = 0$ and prove that if $\Pi \rhd \Gamma \vdash_{\mathtt{B}} (\lambda \mathtt{x.M})\mathtt{N} : \sigma$, then there exists $\Pi' \rhd \Gamma \vdash_{\mathtt{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}]\sigma$ with $\mathbf{rk}(\Pi) \geq \mathbf{rk}(\Pi')$ such that for $r \geq \mathbf{rk}(\Pi)$:

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

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

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

where $\Gamma_1 \# \Gamma_2$, $\Gamma =!^n\Gamma_1, !^n\Gamma_2$, $\sigma \equiv !^n A$ and $n \geq 0$. Clearly, by definition of the space weight δ, we have $\delta(\Pi, r) = r^n(\delta(\Pi_1, r) + 1 + \delta(\Pi_2, r))$. By Lemma 58 there exists a derivation $\Pi_3 \rhd \Gamma \vdash_{\mathtt{B}} \mathtt{M}[\mathtt{N}/\mathtt{x}] : A$ such that $\delta(\Pi_3, r) \leq \delta(\Pi_1, r) + \delta(\Pi_2, r)$. Hence, we can construct Π' ending as:

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

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

2. It follows directly by the definition of the space weight 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 60. *Let* $\Pi \rhd \vdash_{\mathtt{B}} \mathtt{M} : \mathbf{B}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$ *and for each* $r \geq \mathbf{rk}(\Pi)$*:*

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

Proof. It follows directly by Lemma 59. □

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

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

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

Proof. By Lemma 58 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* $\Pi \rhd\vdash_{\mathrm{B}} \mathtt{M} : \mathbf{B}$ *and* $\nabla ::\models \mathtt{M} \Downarrow \mathtt{b}$. *For each configuration* $\phi \succ \mathcal{C}, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b} \in \nabla$ *such that* $\mathcal{C} \not\equiv \circ$ *there exist derivations* $\Sigma \rhd\vdash_{\mathrm{B}} (\mathcal{C}[\mathtt{N}])^{\mathcal{A}} : \mathbf{B}$ *and* $\Theta \rhd\vdash_{\mathrm{B}} (\mathtt{N})^{\mathcal{A}} : \mathbf{B}$ *such that* Θ *is a proper subderivation of* Σ *and for each* $r \geq \mathbf{rk}(\Pi)$*:*

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

5.4.2. Proof of PSPACE Soundness

As stressed 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 occurrences 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 we obtain a bound on the number of applications of the (h) rule in a path.

Definition 34. *The number of* sliced occurrences $n_{so}(\mathtt{x}, \mathtt{M})$ *of the variable* $\mathtt{x}$ *in* $\mathtt{M}$ *is defined as:*

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

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

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

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

Lemma 61. *Let* $\Pi \rhd \Gamma, \mathtt{x} :!^n A \vdash_{\mathrm{B}} \mathtt{M} : \sigma$ *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 $(\mathbf{B}E)$ 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:

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M}' : \mathbf{B} \quad \Theta_{\mathtt{0}} \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_{\mathtt{0}} : B \quad \Theta_1 \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{N}_1 : B}{\Gamma \vdash_{\mathrm{B}} \ \mathtt{if}\ \mathtt{M}'\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1 \ : B}\ (\mathbf{B}E)$$

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 $i \in \{\mathtt{0}, 1\}$. By definition of rank $\mathtt{rk}(\Pi) = \max\{\mathtt{rk}(\Sigma), \mathtt{rk}(\Theta_{\mathtt{0}}), \mathtt{rk}(\Theta_1)\}$ and since by definition $n_{so}(\mathtt{x},\ \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{N}_{\mathtt{0}}\ \mathtt{else}\ \mathtt{N}_1\ )$ is equal to $\max\{n_{so}(\mathtt{x}, \mathtt{M}), n_{so}(\mathtt{x}, \mathtt{N}_{\mathtt{0}}), n_{so}(\mathtt{x}, \mathtt{N}_1)\}$, then the conclusion follows.
Let the last rule of Π be:

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

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 typing gives also dynamical informations about the number of sliced occurrences of a variable.

Lemma 62. *Let* $\Pi \rhd \Gamma, \mathtt{x} :!^n A \vdash_{\mathrm{B}} \mathtt{M} : \sigma$ *and* $\mathtt{M} \to_{\beta\delta} \mathtt{N}$. *Then* $n_{so}(\mathtt{x}, \mathtt{N}) \leq \mathtt{rk}(\Pi)^n$.

Proof. By Lemma 51 and Lemma 61. □

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

Lemma 63. *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 $\Pi \rhd \mathtt{x} :!^n A \vdash_{\mathrm{B}} \mathtt{P} : B$.
By Lemma 62 for every $\mathtt{P}'$ such that $\mathtt{P} \to^*_{\beta\delta} \mathtt{P}'$ 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 of a derivation with both the size of the subject of the conclusion and the degree of the derivation itself.

Lemma 64. *Let* $\Pi \rhd \Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma$.

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

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 δ.
2. 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 δ and d.
3. By definition of rank it is easy to verify that $\mathbf{rk}(\Pi) \leq |\mathtt{M}|$, 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 $\mathbf{B}$-context.

Lemma 65. *Let* $\mathtt{M} \in \mathcal{P}_d$ *and* $\nabla :: \models \mathtt{M} \Downarrow \mathtt{b}$. *Then for each* $\phi \succ C, \mathcal{A} \models \mathtt{N} \Downarrow \mathtt{b}' \in \nabla$*:*

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

Proof.

1. By Lemma 56.1, Lemma 60 and Lemma 64.3.
2. By Lemma 56.2, Lemma 63, Lemma 55.1, Lemma 60 and Lemma 64.3:
$$|\mathtt{N}| \leq (\#_h(\phi) + 1)|\mathtt{M}| \leq \#(\mathcal{A})|\mathtt{M}|^{d+1} + |\mathtt{M}| \leq 2|\mathtt{M}|^{2d+2}$$
3. By Lemma 56.3, the previous point of this lemma, Lemma 55.2, Lemma 60 and Lemma 64.3:
$$|C| \leq \#(C)2|\mathtt{M}|^{2d+2} \leq |\mathtt{M}|^{d+1}2|\mathtt{M}|^{2d+2} \leq 2|\mathtt{M}|^{3d+3}$$

□

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

Theorem 23 (Polynomial Space Soundness).
Let $\mathtt{M} \in \mathcal{P}_d$. *Then:*

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

Proof. By definition of $\mathsf{space}(\mathtt{M})$ and Lemma 65. □

5.5. PSPACE completeness

It is well known that the class of problems decidable by Deterministic Turing machines (DTM) in space polynomial in the length of the input coincides with the class of problems decidable by Alternating Turing machines (ATM) [Chandra et al., 1981] in time polynomial in the length of the input.
In Section 2.4.2 we have shown that polynomial time DTM are definable by λ-terms typable in STA. Analogously here we show that polynomial time 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 consider the same representation of data types as in STA, in particular data types are typable through derivations with degree 0. Finally we show that for each polynomial time ATM $\mathcal{M}$ we can define a recursive evaluation procedure which behaves as $\mathcal{M}$. In what follows we use some of the notation introduced in Section 2.4.2.

Programmable functions The polynomial time completeness in Chapter 2 relies on the notion of λ-*definability*, given in [Barendregt, 1984], 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 also need the following definition.

Definition 35. *Let* $f : \mathbb{I}_1 \times \ldots \times \mathbb{I}_n \to \mathbb{B}$ *and let elements in* $\mathbb{I}_j$ *be representable by terms* $\mathtt{i}_j (1 \leq j \leq n)$.
f *is* programmable *if, for a term* $\mathtt{f} \in \Lambda_{\mathcal{B}}$, $\mathtt{f}\mathtt{i}_1 \ldots \mathtt{i}_n \in \mathcal{P}$ *and:*

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

Tensor Product In Section 2.4.2, we have seen that the tensor product $\otimes$ is definable in STA. In particular by Lemma 46 it is definable in $\mathrm{STA}_{\mathbf{B}}$. Moreover, since $\mathrm{STA}_{\mathbf{B}}$ is an affine system, analogously to STA, tensor product enjoys some properties of the additive conjunction, as to allow the projectors. So in particular we have terms:

$$\pi_1(\mathtt{M}) \doteq \mathtt{M}(\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}) \qquad \pi_2(\mathtt{M}) \doteq \mathtt{M}(\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y})$$

and the following are derived rules.

$$\frac{\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma \otimes \tau}{\Gamma \vdash_{\mathrm{B}} \pi_1(\mathtt{M}) : \sigma} \qquad \frac{\Gamma \vdash_{\mathrm{B}} \mathtt{M} : \sigma \otimes \tau}{\Gamma \vdash_{\mathrm{B}} \pi_2(\mathtt{M}) : \tau}$$

Natural numbers In $\mathrm{STA}_{\mathbf{B}}$, analogously to what happens in STA, natural numbers are represented by Church numerals, i.e. $\mathtt{n} \doteq \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}^n(\mathtt{z})$. As shown in Section 2.4.2 Church numerals and some of the operation usually definable over them are typable by indexed types:

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

In particular for $\mathrm{STA}_{\mathbf{B}}$ it holds the following analogous of Lemma 18.

Lemma 66. *Let P be a polynomial in the variable X and $deg(P)$ its degree. Then there is a term $\underline{\mathtt{P}}$ defining P typable as :*

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

Proof. Obvious by Lemma 18 and Lemma 46. □

Boolean connectives Clearly since basic booleans constants are primitive in $\mathrm{STA}_{\mathbf{B}}$ we can define easily the usual boolean connectives. In particular we have the following terms:

$$\begin{aligned} \mathtt{M}\ \underline{\mathtt{and}}\ \mathtt{N} &\doteq \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ \mathtt{1}\ )\ \mathtt{else}\ \mathtt{1} \\ \mathtt{M}\ \underline{\mathtt{or}}\ \mathtt{N} &\doteq \mathtt{if}\ \mathtt{M}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ (\ \mathtt{if}\ \mathtt{N}\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ \mathtt{1}\ ) \end{aligned}$$

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

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

ATM Configurations The encoding of Deterministic Turing machine configuration given in Section 2.4.2 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* ($\mathtt{A}$), *rejecting* ($\mathtt{R}$), *universal* ($\wedge$), *existential* ($\vee$) . We can encode such information by tensor pairs of booleans. In particular:

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

We say that a configuration is accepting, rejecting, universal or existential depending on the kind of its state.
Let $\circ$ denote composition as defined in Section 2.4.2. Then, we can encode ATM configurations by terms of the shape:

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

where $\mathsf{cb}_0^l \circ \ldots \circ \mathsf{cb}_n^l$ and $\mathsf{cb}_0^r \circ \ldots \circ \mathsf{cb}_n^r$ are respectively the left and right hand-side words on the ATM tape, $\mathtt{Q}$ is a tuple 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. Analogously to the case of Deterministic Turing machines, by convention the left part of the tape is represented in a reversed order, the alphabet is composed by the two symbols $\mathtt{0}$ and 1, the scanned symbol is the first symbol in the right part and final states are divided in accepting and rejecting.

Definition 36. *The* indexed type $\mathbf{ATM}_i^q$ *for each* $i, q \in \mathbb{N}$ *is defined as:*

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

The above indexed type is useful to type ATM configurations.

Lemma 67. *Let* $t \in \mathbb{N}$. *Every term of the shape:*

$$\lambda c.\langle \mathsf{cb}_0^l \circ \cdots \circ \mathsf{cb}_n^l, \mathsf{cb}_0^r \circ \cdots \circ \mathsf{cb}_m^r, \langle \mathtt{q}_0, \ldots, \mathtt{q}_t, \mathtt{k}_1, \mathtt{k}_2\rangle\rangle$$

defines an ATM configuration. For every $i > 0$ *such terms are typable in* $\mathrm{STA}_{\mathbf{B}}$ *as:*

$$\vdash_{\mathrm{B}} \lambda c.\langle \mathsf{cb}_0^l \circ \cdots \circ \mathsf{cb}_n^l, \mathsf{cb}_0^r \circ \cdots \circ \mathsf{cb}_m^r, \langle \mathtt{q}_0, \ldots, \mathtt{q}_t, \mathtt{k}_1, \mathtt{k}_2\rangle\rangle : \mathbf{ATM}_i^n$$

Proof. Easy. □

It is easy to adapt the terms described in 2.4.2 dealing with TM to the case of ATM.

Lemma 68. *The term* $\mathbf{Init}_A \doteq \lambda t.\lambda c.\langle \lambda z.z, \lambda z.t(c\mathbf{0})z, \langle \underline{Q_0}, k_0 \rangle \rangle$ *defines the function that, taking as input a Church numeral* $\underline{n}$*, gives as output an alternating Turing machine with tape of length* n *filled by* $\mathbf{0}$*'s in the initial state* $Q_0 \equiv \langle q_0, \ldots, q_n \rangle$*, with initial kind* $k_0 \equiv \langle k_0', k_0'' \rangle$ *and with the head at the beginning of the tape. For each* $i \in \mathbf{N}$ *it is typable in* STA *as:*

$$\vdash_B \mathbf{Init}_A : \mathbf{N}_i \multimap \mathbf{TM}_i^n$$

Proof. Easy. □

While DTM behaviour is determined by a *transition function*, ATM behaviour is instead determined by a *transition relation*. Nevertheless we can regard a transition relation as the composition of different transition functions. So we can consider only the latter.

Lemma 69. *For every transition function* f *between ATM configurations, there exists a term* $\mathbf{Tr}_f$ *defining an ATM transition step typable as:*

$$\vdash_B \mathbf{Tr}_f : \mathbf{ATM}_i \multimap \mathbf{ATM}_i$$

Proof. Analogous to the proof of Lemma 27. □

As in the case of Deterministic Turing machines we need a term that initialize an Alternating Turing machine with an input string.

Lemma 70. *There exists a term* $\mathbf{In}_A$ *defining the function that, when supplied by a boolean string and an ATM, writes the input string on the tape of the ATM. Such a term is typable as:*

$$\vdash_B \mathbf{In}_A : \mathbf{S} \multimap \mathbf{ATM}_i^q \multimap \mathbf{ATM}_i^q$$

Proof. Similar to the proof of Lemma 28. □

In what follows, we need a terms that returns the kind of a given configuration.

Lemma 71. *The term:*

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

defines the function returning the kind of an input configuration. It is typable in $\mathrm{STA}_{\mathbf{B}}$ *as:*

$$\vdash_B \mathbf{Kind} : \mathbf{ATM}_i \multimap \mathbf{B}^2$$

Proof. Easy. □

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

Lemma 72. *The term:*

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

defines the function that given an ATM configuration returns $\mathbf{0}$ *if it is accepting,* $\mathbf{1}$ *otherwise. It is typable in* $\mathrm{STA}_{\mathbf{B}}$ *as:*

$$\vdash_{\mathtt{B}} \mathtt{Ext} : \mathbf{ATM}_{\mathtt{i}} \multimap \mathbf{B}$$

Proof. Easy. □

Evaluation function Given an ATM $\mathcal{M}$ working in polynomial time we define a recursive evaluation procedure $\mathbf{eval}_{\mathcal{M}}$ which takes a string $\mathtt{s}$ and returns $\mathtt{0}$ or $\mathtt{1}$ if the initial configuration (with the tape filled with $\mathtt{s}$) leads to an accepting or rejecting configuration respectively.
Without loss of generality we consider ATMs with transition relation of degree two (at each step we consider two transitions). We need to define some auxiliary functions.

Lemma 73. *The term:*

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

defines a function α *acting as:*

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

It is typable in $\mathrm{STA}_{\mathbf{B}}$ *by the following typing rule:*

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

where the management of contexts is additive.

Proof. Checking that $\underline{\alpha}$ has the intended behaviour and typing is boring but easy. □

We would now define $\mathbf{eval}_{\mathcal{M}}$ as an iteration of an higher order $\mathbf{Step}_{\mathcal{M}}$ function over a $\mathbf{Base}$ case.

Lemma 74. *Let* $\mathbf{Tr}_1$ *and* $\mathbf{Tr}_2$ *be two closed terms defining the two components of the transition relation. The terms:*

$$\mathbf{Base} \doteq \lambda \mathtt{c}.(\mathbf{Kind}\ \mathtt{c}) \quad \textit{and} \quad \mathbf{Step}_{\mathcal{M}} \doteq \lambda \mathtt{h}.\lambda \mathtt{c}.\underline{\alpha}((\mathbf{Kind}\ \mathtt{c}), (\mathtt{h}(\mathbf{Tr}_1\ \mathtt{c})), (\mathtt{h}(\mathbf{Tr}_2\ \mathtt{c})))$$

are typable respectively as:

$$\vdash_{\mathrm{B}} \mathbf{Base} : \mathbf{ATM}_i \multimap \mathbf{B}^2 \quad \textit{and} \quad \vdash_{\mathrm{B}} \mathbf{Step}_{\mathcal{M}} : (\mathbf{ATM}_i \multimap \mathbf{B}^2) \multimap \mathbf{ATM}_i \multimap \mathbf{B}^2$$

Proof. Easy. □

Now we can finally define the evaluation function.

Lemma 75. *Let* P *be a polynomial. Then, the term:*

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

defines the evaluation function of an ATM $\mathcal{M}$ *working in time* $P(|s|)$*, for every input string* s*. It is typable in* $\mathrm{STA}_{\mathbf{B}}$ *as*

$$\vdash_{\mathrm{B}} \mathbf{eval}_{\mathcal{M}} : !^{\max(deg(P),1)+1}\mathbf{S} \multimap \mathbf{B}$$

Proof. Checking that $\underline{\alpha}$ has the intended behaviour and typing is boring but easy. In particular, it follows by Lemma 66 using Lemma 68, Lemma 69, Lemma 70, Lemma 73, Lemma 74 and Lemma 72. □

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 76. *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. By Lemma 75 it follows:

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

□

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

Theorem 24 (Polynomial Space Completeness). *Every decision problem* $\mathcal{D} \in$ PSPACE *is programmable in* $\mathrm{STA}_{\mathbf{B}}$*.*

5.6. Related Topic

In this section we briefly discuss some topic which are related to the topics presented in the previous sections.

FPSPACE characterization. FPSPACE is the class of functions 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

$$\frac{\Gamma \vdash \mathtt{M} : \mathbf{W} \quad \Gamma \vdash \mathtt{N}_\epsilon : \sigma \quad \Gamma \vdash \mathtt{N_0} : \sigma \quad \Gamma \vdash \mathtt{N_1} : \sigma}{\Gamma \vdash \mathbf{D}(\mathtt{M}, \mathtt{N}_\epsilon, \mathtt{N_0}, \mathtt{N_1}) : \sigma}$$

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 [Leivant and Marion, 1993], completeness for FPSPACE can be proved by considering two distinct data types $\mathbf{S}$ (Church representations of Strings) and $\mathbf{W}$ (Flat words over Booleans) as input and output data type respectively. 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 [Lafont, 2004]. $\mathrm{STA}_{\mathbf{B}}$ 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 [Ronchi Della Rocca and Roversi, 1997] are:

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

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:

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

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 mimic in the cut elimination process the abstract machine mechanism of Section 5.3. For this reason it could be more interesting introduce proof-nets for such a system and study cut elimination in this framework.

Part II.

Linearity and Semantics

This part of the thesis studies the problem of designing a programming language *for computable functions* which is, in a sense that will later on become clear, *semantically linear*.

The Programming language for Computable Functions PCF was introduced by Plotkin in [Plotkin, 1977] as the programming language counterpart of Scott's Logic for Computable Functions LCF [Scott, 1993]. The very name PCF stresses the fact that it is a programming language where all the computable functions can be programmed. PCF is in fact a paradigmatic example of a typed functional programming language. It consists of the simply typed lambda calculus augmented by basic arithmetic constants and by general recursion in the form of fixpoint combinators at every type.

PCF is simple and general. For these reasons, during the years, it has been widely used as base language for semantical investigations. In particular, PCF has been at the centre of the research on relations between operational and denotational semantics. See [Ong, 1995] for an extensive treatment of the subject and [Curien, 2007] for a more recent survey.

The operational semantics describe the meaning of a program in terms of the observable result of its execution by an abstract mathematical machine. For this reason two programs are considered operationally equivalent whenever they are interchangeable "in all contexts" without affecting the observable outcome of the computation. Denotational semantics, on the other hand, describes the meaning of a program in terms of its interpretation in a mathematical model. Two programs are denotationally equivalent in a given model only when they are interpreted in the same object of the model.

A model is said fully abstract with respect to an operational semantics if the denotational equivalence induced by the model and the operational equivalence induced by the operational semantics coincide.

The quest for a fully abstract model for PCF has influenced the researches in the study of programming language semantics from the very beginning. In particular, as stressed in [Curien, 2007], we can distinguish between two approaches in the investigations related to this problem. The first approach is to look for a model fitting an intended fixed language. Many works following this approach have led to the discovery of different relevant semantical models. Among the most important, dI Domains [Berry, 1978], coherence spaces [Girard, 1987], and game semantics, see [Hyland and Ong, 2000] for a survey.

The second complementary approach is to look for a language fitting an intended model. The content of this part of the thesis follows this second approach. The aim is in fact to design a programming language for which a specific model is fully abstract.

The model we are interested in is the fragment of Girard's coherence spaces including the infinite flat domain $\mathbf{N}$, representing natural numbers, and the linear function space constructor $\multimap$, as the only coherence space constructor.
It has been proved in [Plotkin, 1977], for the sake of varying the language to fit a model, that Scott-continuous domains are fully abstract and universal for PCF^{++}, namely PCF extended with a parallel conditional `pif` and an existential operator $\exists$. Moreover, Paolini has shown that stable domains are fully abstract for StPCF [Paolini, 2006], an extension of PCF. The language StPCF is obtained by extending PCF with two operators: `gor` and `strict?`. `gor` corresponds to a Gustave-like `or` function, while `strict?` corresponds to a non-extensional, monotone function.
Linear functions are both continuous and stable, and they are both extensionally and stably ordered. Moreover, they are strict in all their arguments. Consequently, the language fully abstract for linear functions we are looking for must be, in some sense, a restriction of the "intersection" of PCF^{++} and StPCF where programs are strict in all their arguments. Nonetheless, `strict?` does not respect the extensional order, while `pif` and $\exists$ do not respect the stable order. For this reason, they cannot belong to our language. `gor` respects both extensional and stable orders, but it is not strict, and therefore not linear. As a result, it is natural to ask if a restriction of PCF is sufficient in order to gain the full abstraction with respect to the linear model, or if it is necessary to extend PCF with operators capturing the linear kernel of some operators above.

This framework led us to formalize a new language, named $\mathcal{SL}$PCF. The acronym stands for $\mathcal{S}$emantically ℓinear Programming language for Computable Functions. The $\mathcal{SL}$PCF language, being a restriction of PCF, includes λ-abstraction and application, $\mathtt{s}, \mathtt{0}$ and `p` constants for the successor, the zero and predecessor functions respectively, an `if` conditional, and a μ-abstraction for fixpoint definitions. Specific to $\mathcal{SL}$PCF is the distinction between different kinds of variables: higher-order, ground and stable. Higher-order variables must obey to some constraints, which are necessary in order to respect linearity and strictness. Ground variables, on the other hand, can be freely used. The strictness for functions with ground arguments is provided by the operational semantics. Finally, stable variables are used only in fixpoint definitions, hence they can only be μ-abstracted. Linearity and strictness are again assured by constraints on terms formation.
The operational semantics of $\mathcal{SL}$PCF, described through an abstract machine in the style of a big-step Structural Operational Semantics [Plotkin, 2004, Kahn, 1987], mixes call-by-name and call-by-value parameter passing. In particular, the parameter passing

mechanism for the evaluation of redexes abstracting higher-order variables is call-by-name, while the one for the evaluation of redexes abstracting ground variables is call-by-value.

$\mathcal{SL}$PCF can be interpreted in linear coherence space and we show this interpretation is correct with respect to the operational semantics. Correctness implies that $\mathcal{SL}$PCF captures some of the key points behind the linear coherence spaces model.

We show that all Kleene-recursive functions can be programmed in our language. Therefore, in some sense we obtain the first-order full abstraction as a corollary result.

Nonetheless, the interpretation in the linear coherence spaces model is not fully abstract with respect to the operational semantics. In particular, there are finite cliques in linear coherence spaces which have no counterpart in the language, hence definability fails.

It should be stressed here that the $\mathcal{SL}$PCF language presented in this part of the thesis is a first step toward a programming language for which the interpretation in the linear coherence spaces model is fully abstract. We conclude by suggesting a possible extension.

$\mathcal{SL}$PCF can be compared to languages appeared in literature. Even if with different goals, in [Alves et al., 2006] the authors propose a language whose treatment of conditional is technically similar to the one in $\mathcal{SL}$PCF. A use of ground variables analogous to the one in $\mathcal{SL}$PCF can be found in the language proposed in [Bellantoni et al., 2000] with the goal of implicitly characterizing complexity classes. Finally, in [Bierman et al., 2000] a language with a fixpoint operator in a linear framework is studied. The authors make a distinction between intuitionistic and linear variables which is similar to the distinction in $\mathcal{SL}$PCF between stable and linear variables. Nevertheless, none of these works put all these things together.

The content of this chapter is an extended version of the work presented in [Gaboardi and Paolini, 2007].

6. A Semantically Linear Programming Language for Computable Functions

6.1. Introduction

In this chapter we study the language $\mathcal{SL}\mathrm{PCF}$, a syntactical restriction of PCF. The main result of this chapter is the proof that $\mathcal{SL}\mathrm{PCF}$ is able to program only functions of a purely linear model. We start by recalling the notion of linear coherence spaces, we present the syntax of $\mathcal{SL}\mathrm{PCF}$ and describe its operational semantics. Then we define the denotational interpretation of the language in linear coherence spaces and we prove that such interpretation is correct. We show that all partial recursive function can be defined in $\mathcal{SL}\mathrm{PCF}$. We conclude by showing that $\mathcal{SL}\mathrm{PCF}$ is not complete (and so not fully abstract) with respect to linear coherence spaces hence we describe the problem connected with some possible extensions.

6.2. Linear Coherence Spaces

In this section we recall some notions about coherence spaces. We recall the notions of continuous, stable and linear functions and we show how linear function can be represented as a coherence space. We introduce two orders, the extensional and the stable one, on stable and linear functions and we show their relations. Finally we analyze the flat coherence space $\mathbf{N}$ of natural numbers.

6.2.1. Coherence spaces

Coherence spaces are a simple framework for Berry's stable functions [Berry, 1978], developed by Girard [Girard, 1987].

Definition 37. *A* coherence space X *is a pair* $\langle |X|, \frown_X \rangle$ *where* $|X|$ *is a set of* tokens *called the* web *of* X *and* $\frown_X$ *is a reflexive and transitive relation between tokens of* $|X|$ *called the* coherence relation *on* X*. A clique* x *of* X *is a subset of* $|X|$ *made of pairwise coherent tokens. The set of cliques of* X *is denoted* $Cl(X)$*.*

The strict incoherence $\smile_X$ is the complementary relation of $\frown_X$; the incoherence $\asymp_X$ is the union of relations $\smile_X$ and $=$; the strict coherence $\frown_X$ is the complementary relation of $\asymp_X$.

Coherence spaces are ranged over by X, Y, Z, tokens are ranged over by a, b, c while cliques by x, y, z. When there is no ambiguity we use X both to denote the coherence space X and its web $|X|$. Cliques are sets, hence we use the standard set notation for them.
Recall that a set D is directed if and only if it is non empty and for every $x, y \in D$ there exists $z \in D$ such that $x, y \subseteq z$. If X is a coherence space then $Cl(X)$ form a complete partial order with respect to the set-theoretical inclusion. In particular:

Lemma 77. *Let X be a coherence space. Then:*

1. *$\emptyset \in Cl(X)$ and $\{a\} \in Cl(X)$, for each $a \in |X|$,*
2. *if $y \subseteq x$ and $x \in Cl(X)$ then $y \in Cl(X)$,*
3. *if $D \subseteq Cl(X)$ is directed then $\bigcup D \in Cl(X)$.*

Proof.

1. Easy by definition of $Cl(X)$.
2. Since $\forall a, b \in x:\ a \frown b$ then $\forall a, b \in x \subseteq y:\ a \frown b$.
3. Consider generic $a, b \in \bigcup D$. Clearly there exist $x, y \in D$ such that $a \in x$ and $b \in y$. Since D is directed there exists $z \in D \subset Cl(X)$ such that $a, b \in z$ but this implies $a \frown_X b$ hence the conclusion follows. □

We are interested in studying functions over cliques.

Definition 38. *Let X and Y be coherence spaces and $f : Cl(X) \to Cl(Y)$ be a monotone function.*

- *f is* continuous *whenever $\forall x \in Cl(X)$, $\forall a \in f(x)$, $\exists x_0 \subseteq_{fin} x$ such that $a \in f(x_0)$.*
- *f is* stable *whenever $\forall x \in Cl(X)$, $\forall a \in f(x)$, $\exists x_0 \subseteq_{fin} x$ such that $a \in f(x_0)$ and $\forall x' \subseteq x$, if $a \in f(x')$ then $x_0 \subseteq x'$.*
- *f is* linear *whenever $\forall x \in Cl(X)$, $\forall a \in f(x)$, $\exists b \in x$ such that $a \in f(\{b\})$.*

Using the above definition it is easy to verify the following lemma.

Lemma 78. *Let X and Y be coherence spaces.*

1. *Every stable function $f : Cl(X) \to Cl(Y)$ is also continuous.*
2. *Every linear function $f : Cl(X) \to Cl(Y)$ is also stable.*

Proof. Easy. □

The following lemma shows that linear functions are all strict.

Lemma 79. *If $f : Cl(X) \to Cl(X)$ is a linear function, then $f(\emptyset) = \emptyset$.*

Proof. If $f(\emptyset) = x$ for some $x \neq \emptyset \in Cl(X)$ then by definition of linear function for each $a \in x$ there is $b \in \emptyset$ such that $a \in f(\{b\})$. But clearly this is an absurd hence x must be equal to $\emptyset$. □

Continuous, stable and linear function can be characterized in a different way.

Lemma 80. *Let X and Y be coherence spaces.*

1. *If $f : Cl(X) \to Cl(Y)$ is a monotone function then f is continuous if and only if $f(\bigcup D) = \bigcup\{f(x) \mid x \in D\}$, for each directed $D \subseteq Cl(X)$.*
2. *If $f : Cl(X) \to Cl(Y)$ is a continuous function then f is stable if and only if $\forall x, y \in Cl(X)$, $x \cup y \in Cl(X)$ implies $f(x \cap y) = f(x) \cap f(y)$.*
3. *If $f : Cl(X) \to Cl(Y)$ is a stable function then f is linear if and only if $\forall x \in Cl(X) : \ f(x) = \bigcup\{f(\{a\}) \mid a \in x\}$.*

Proof.

1. Suppose f is continuous and consider $D \subseteq X$ directed. Clearly for every $x \in D$: $x \subseteq \bigcup D$ hence by monotonicity $f(x) \subseteq f(\bigcup D)$ and so $\bigcup\{f(x) \mid x \in D\} \subseteq f(\bigcup D)$. Now let $b \in f(\bigcup D)$, by definition of continuous function $\exists x_0 \subseteq_{fin} \bigcup D$ such that $b \in f(x_0)$. This implies that for every $a_i \in x_0$ there exists $y_i \in D$ such that $a_i \in y_i$. Since D is directed there exist $y \in D$ such that $y_i \subseteq y$. So $x_0 \subseteq y$ and by monotonicity $f(x_0) \subseteq f(y)$. Hence $b \in f(y)$ and we can conclude $b \in \bigcup\{f(x) \mid x \in D\}$. Conversely consider $x \in Cl(X)$ and $b \in f(x)$. The set $\{x_i \mid x_i \subseteq_{fin} x\}$ is directed and $\bigcup\{x_i \mid x_i \subseteq_{fin} x\} = x$, so $f(x) = \bigcup\{f(x_i) \mid x_i \subseteq_{fin} x\}$. Now $b \in \bigcup\{f(x_i) \mid x_i \subseteq_{fin} x\}$ implies that there exists $x_i \subseteq_{fin} x$ such that $b \in f(x_i)$ and so the conclusion follows.

2. Suppose f stable and consider $x, y \in Cl(X)$ such that $x \cup y \in Cl(X)$. Since $x \cap y \subseteq x$ then by monotonicity $f(x \cap y) \subseteq f(x)$ and analogously $f(x \cap y) \subseteq f(y)$. Hence $f(x \cap y) \subseteq f(x) \cap f(y)$. Now consider $a \in f(x) \cap f(y)$, then $a \in f(x)$ and $a \in f(y)$ and since $x \cup y \in Cl(X)$ by monotonicity $a \in f(x \cup y)$. Since f is stable there exists $x_0 \subseteq_{fin} x \cup y$ such that $a \in f(x_0)$ and $\forall x' \subseteq x \cup y$, if $a \in f(x')$ then $x_0 \subseteq x'$. So in particular $x_0 \subseteq x$ and $x_0 \subseteq y$ but hence $x_0 \subseteq x \cap y$ and so $a \in f(x \cap y)$.
Conversely consider $x \in Cl(X)$ and $b \in f(x)$. Since f is continuous there exists $x_0 \subseteq_{fin} x$ such that $b \in f(x_0)$. Now take the minimal such x_0 and consider $x' \subseteq x$ such that $b \in f(x')$. So we have $b \in f(x_0) \cap f(x')$ and by hypothesis $b \in f(x_0 \cap x')$. Hence since we have chosen x_0 minimal we have $x_0 \subseteq x_0 \cap x'$ and we can conclude $x_0 \subseteq x'$.

3. Suppose f is linear and consider $x \in Cl(X)$. Clearly for every $b \in x$: $\{b\} \subseteq x$ hence by monotonicity $f(\{b\}) \subseteq f(x)$ and so $\bigcup\{f(\{b\}) \mid b \in x\} \subseteq f(x)$. Now for every $a \in f(x)$, since f is linear, there exists $b \in x$ such that $a \in f(\{b\})$. So for every $a \in f(x)$: $a \in \bigcup\{f(\{b\}) \mid b \in x\}$ and hence $f(x) \subseteq \bigcup\{f(\{b\}) \mid b \in x\}$.
Conversely consider $x \in Cl(X)$ and $a \in f(x)$. Since by hypothesis $f(x) = \bigcup\{f(\{b\}) \mid b \in x\}$ it is immediate that there exists b such that $a \in f(\{b\})$. □

It is well known that every continuous function over a complete partial order has a least fixed point. Let $f^n(\emptyset)$ denotes the iteration n-times of the function f over the empty set recursively defined as $f^{n+1}(\emptyset) = f(f^n(\emptyset))$ and $f^0(\emptyset) = \emptyset$. Then we have the following.

Theorem 25. *Let X be a coherence space. If $f : Cl(X) \to Cl(X)$ is continuous then there exists $\mathtt{fix}(f) = \bigcup f^n(\emptyset) \in Cl(X)$, such that $\mathtt{fix}(f) = f(\mathtt{fix}(f))$ and $\mathtt{fix}(f) \subseteq x$ for any $x \in Cl(X)$ such that $x = f(x)$.*

Proof. Clearly the set $\{f^n(\emptyset) \mid n \in \mathbb{N}\}$ is directed since f is monotone. Hence by Lemma 80: $f(\bigcup f^n(\emptyset)) = \bigcup f^{n+1}(\emptyset) = \bigcup f^n(\emptyset)$.
Now suppose there is $x \in Cl(X)$ such that $x = f(x)$. We prove by induction over $n \in \mathbb{N}$ that $f^n(\emptyset) \subseteq x$. Clearly $\emptyset \subseteq x$. Suppose $f^n(\emptyset) \subseteq x$ then by monotonicity $f^{n+1}(\emptyset) \subseteq f(x) = x$. Hence in particular $\bigcup f^n(\emptyset) \subseteq x$. □

Note that Lemma 79 implies that the least fixed point of a linear function is always $\emptyset$. In what follows it will be useful the following lemma.

Lemma 81. *If $f : Cl(X) \to Cl(X)$ is a linear function, then $\forall x \in Cl(X), \forall a \in f(x)$, $\exists b \in x$ such that $a \in f(\{b\})$ and b is unique.*

Proof. Suppose b is not unique then there exists another $c \in x$ such that $a \in f(\{c\})$. Since $b, c \in x$ by Lemma 77.2 it follows $\{b, c\} \in Cl(X)$. So in particular since f is stable by Lemma 80.2 $a \in f(\{b\}) \cap f(\{c\}) = f(\{b\} \cap \{c\}) = f(\emptyset)$ but this contradicts Lemma 79. □

6.2.2. The linear functions coherence space

We define the linear functions coherence space $X \multimap Y$ from X to Y.

Definition 39. *Let X and Y be coherence spaces. $X \multimap Y$ is the coherence space having $|X \multimap Y| = |X| \times |Y|$ as web, while:*

$$(a, b) \frown_{X \multimap Y} (a', b') \iff a \frown_X a' \text{ implies } b \frown_Y b' \text{ and } a \smile_X a' \text{ implies } b \smile_Y b'$$

In fact $X \multimap Y$ does not corresponds to the set of linear functions from X to Y. Nevertheless, we can give a different representation of linear functions.

Definition 40. *Let X and Y be coherence spaces. The* trace *of a linear function $f : Cl(X) \to Cl(Y)$ is the set defined as follows:*

$$\mathcal{Tr}(f) = \{(a, b) \in |X| \times |Y| \mid b \in f(\{a\})\}$$

Lemma 82. *If $f : Cl(X) \to Cl(Y)$ is a linear function then $\mathcal{Tr}(f) \in Cl(X \multimap Y)$.*

Proof. Let $(a, b), (a', b') \in \mathcal{Tr}(f)$, then we want to prove that $(a, b) \frown_{X \multimap Y} (a', b')$. Suppose $a \frown_X a'$ then there exists $x \in Cl(X)$ such that $a, a' \in x$. Now by definition of $\mathcal{Tr}(f)$: $b, b' \in f(x)$ hence in particular $b \frown_Y b'$.
Otherwise suppose $a \smile_X a'$. Since $\{a, a'\} \in Cl(X)$ then $b, b' \in f(\{a, a'\})$, if $b = b'$ then by Lemma 81 $a = a'$, but this contradicts the hypothesis $a \smile_X a'$ hence necessarily $b \smile_Y b'$. □

So linear functions can be represented as cliques.

Definition 41. *Let X and Y be coherence spaces, $t \in Cl(X \multimap Y)$ and $x \in Cl(X)$. Let us define the map $\mathcal{F}(t) : Cl(X) \to Cl(Y)$ to be the function such that*

$$\mathcal{F}(t)(x) = \{b \in |Y| \mid \exists a \in x, (a, b) \in t\}$$

Lemma 83. *If $t \in Cl(X \multimap Y)$ then $\mathcal{F}(t) : Cl(X) \to Cl(Y)$ is a linear function.*

Proof. Clearly $\mathcal{F}(t)$ is monotone. In order to prove that it is also linear we use the characterizations of Lemma 80.
Consider $D \subset Cl(X)$ directed. Clearly for every $x \in D$: $x \subseteq \bigcup D$ hence by monotonicity $\mathcal{F}(t)(x) \subseteq \mathcal{F}(t)(\bigcup D)$ and so $\bigcup\{\mathcal{F}(t)(x) \mid x \in D\} \subseteq \mathcal{F}(t)(\bigcup D)$. Now let $b \in \mathcal{F}(t)(\bigcup D)$, hence there exists $a \in \bigcup D$ such that $(a, b) \in t$ but hence in particular $b \in \mathcal{F}(t)(\{a\})$ and $b \in \bigcup\{\mathcal{F}(t)(x) \mid x \in D\}$. So $\mathcal{F}(t)$ is continuous.
Now consider $x, y \in Cl(X)$ such that $x \cup y \in Cl(X)$. Since $x \cap y \subseteq x$ then by monotonicity $\mathcal{F}(t)(x \cap y) \subseteq \mathcal{F}(t)(x)$ and analogously $\mathcal{F}(t)(x \cap y) \subseteq \mathcal{F}(t)(y)$. Hence $\mathcal{F}(t)(x \cap y) \subseteq \mathcal{F}(t)(x) \cap \mathcal{F}(t)(y)$. Suppose $b \in \mathcal{F}(t)(x) \cap \mathcal{F}(t)(y)$, then $b \in \mathcal{F}(t)(x)$ and $b \in \mathcal{F}(t)(y)$, so in particular there exists $a \in x$ and $a' \in y$ such that $(a, b), (a', b) \in t$. This implies $(a, b) \frown (a', b)$ and since $\{a, a'\} \in x \cup y \in Cl(X)$ it follows $a = a'$. Hence $a \in x \cap y$ and then $b \in \mathcal{F}(t)(x \cap y)$. So $\mathcal{F}(t)$ is also stable.
Finally consider $x \in Cl(X)$. Clearly for every $b \in x$: $\{b\} \subseteq x$ hence by monotonicity $\mathcal{F}(t)(\{b\}) \subseteq \mathcal{F}(t)(x)$ and so $\bigcup\{\mathcal{F}(t)(\{b\}) \mid b \in x\} \subseteq \mathcal{F}(t)(x)$. Now suppose $a \in \mathcal{F}(t)(x)$ then there exists $b \in x$ such that $(b, a) \in t$ but this implies that $a \in \mathcal{F}(t)(\{b\})$ hence $a \in \bigcup\{\mathcal{F}(t)(\{b\}) \mid b \in x\}$. So $\mathcal{F}(t)$ is linear. □

So every clique of the coherence space $X \multimap Y$ is the representation of a linear function.

Lemma 84. *Let $f : Cl(X) \to Cl(Y)$ be a linear function. Then:*

$$(a, b), (a', b) \in \mathcal{T}r(f) \implies a = a'$$

Proof. By Lemma 82 $\mathcal{T}r(f) \in Cl(X \multimap Y)$ hence in particular $\{(a, b), (a', b)\} \in Cl(X \multimap Y)$. By Lemma 83 $\mathcal{F}(\{(a, b), (a', b)\})$ is a linear function such that $b \in \mathcal{F}(\{(a, b), (a', b)\})(a)$ and $b \in \mathcal{F}(\{(a, b), (a', b)\})(a')$ hence by Lemma 81 $a = a'$. □

Functions between coherence spaces can be ordered in different ways. In particular we are interested in two of them which can be defined over stable functions.

Definition 42. *Let $f, g : Cl(X) \to Cl(Y)$ be stable functions.*

- *f and g are* extensionally ordered, *in symbols $f \sqsubseteq_E g$, if for every $x \in Cl(X)$:*

$$f(x) \subseteq g(x)$$

- *f and g are* stably ordered, *in symbols $f \sqsubseteq_S g$, if for every $x, z \in Cl(X)$:*

$$x \subseteq z \text{ implies } f(x) = f(z) \cap g(x)$$

It is easy to verify that $\sqsubseteq_S$ and $\sqsubseteq_E$ are effectively preorders. The $\sqsubseteq_S$ order corresponds to the inclusion between traces.

Lemma 85. *Let $f, g : Cl(X) \to Cl(Y)$ be linear functions. Then:*

$$f \sqsubseteq_S g \iff \mathcal{T}r(f) \subseteq \mathcal{T}r(g)$$

Proof. Suppose $f \sqsubseteq_S g$ and $(a, b) \in \mathcal{T}r(f)$. Clearly $f(\{a\}) \subseteq g(\{a\})$, since $f(\{a\}) = f(\{a\}) \cap g(\{a\})$. So in particular $b \in g(\{a\})$ hence $(a, b) \in \mathcal{T}r(g)$.
Conversely suppose $\mathcal{T}r(f) \subseteq \mathcal{T}r(g)$ and $x, z \in Cl(X)$ such that $x \subseteq z$. Then, since $f(x) \subseteq g(x)$ for every $x \in Cl(X)$, by monotonicity $f(x) \subseteq f(z) \cap g(x)$. Now consider $b \in f(z) \cap g(x)$ then there exists $a_z \in z$ and $a_x \in x$ such that $b \in f(\{a_z\})$ and $b \in g(\{a_x\})$, hence in particular $(a_z, b) \in \mathcal{T}r(f)$ and $(a_x, b) \in \mathcal{T}r(g)$ and so $(a_z, b), (a_x, b) \in \mathcal{T}r(g)$ since by hypothesis $\mathcal{T}r(f) \subseteq \mathcal{T}r(g)$. By Lemma 84 this implies $a_z = a_x$ hence $b \in f(\{a_x\}) \subseteq f(x)$. □

The following lemma shows that if two stable functions are stably ordered then they are also extensionally ordered.

Lemma 86. *Let $f, g : Cl(X) \to Cl(Y)$ be stable functions. Then:*

$$f \sqsubseteq_S g \implies f \sqsubseteq_E g$$

Proof. Suppose $f \sqsubseteq_S g$. Then for each $x \in Cl(X)$, $f(x) = f(x) \cap g(x)$, thus $f(x) \subseteq g(x)$. □

The converse of the above lemma fails for stable functions. Nevertheless, it holds for linear functions.

Lemma 87. *If $f, g : Cl(X) \to Cl(Y)$ are linear functions then,*

$$f \sqsubseteq_S g \iff f \sqsubseteq_E g.$$

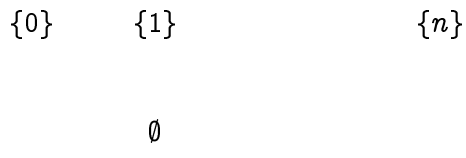
Proof. The if part follows directly by Lemma 78 and Lemma 86. So consider the only if part. Suppose $f \sqsubseteq_E g$ and let $(a, b) \in \mathcal{T}r(f)$. Since by hypothesis $b \in f(\{a\}) \subseteq g(\{a\})$, then $(a, b) \in \mathcal{T}r(g)$. Hence $\mathcal{T}r(f) \subseteq \mathcal{T}r(g)$ and by Lemma 85 the conclusion follows. □

6.2.3. The coherence space $\mathbf{N}$

In the following chapter we will study a model based on the coherence space of natural numbers $\mathbf{N}$.

Definition 43. *Let $\mathbf{N}$ denotes the coherence space of* natural numbers, *namely $(|\mathbf{N}|, \frown_{\mathbf{N}})$ such that $|\mathbf{N}| = \mathbb{N}$ and $m \frown_{\mathbf{N}} n$ if and only if $m = n$, for all $m, n \in |\mathbf{N}|$.*

The set of cliques of $\mathbf{N}$: $Cl(\mathbf{N}) = \{\emptyset\} \cup \{\ \{n\} \mid n \in |\mathbf{N}|\ \}$ endowed with set-theoretical inclusion form the following infinite flat domain:



We are interested in linear functions between natural numbers.

Lemma 88. *For every linear function $f : Cl(\mathbf{N}) \to Cl(\mathbf{N})$ the trace $\mathcal{T}r(f)$ of f has the shape:*

$$\{(n_0, m_0), \ldots, (n_k, m_k) \mid n_0 \neq n_1 \neq \cdots \neq n_k\}$$

for some $n_0, \ldots n_k, m_0, \ldots, m_k \in \mathbb{N}$ and $k \geq 0$.

Proof. By definition $\mathcal{T}r(f)$ is a set of pairs of the shape (n, m) with $n, m \in \mathbb{N}$. Now consider $(n, m), (n', m') \in \mathcal{T}r(f)$ then by Lemma 82 $(n, m) \frown_{\mathbf{N} \multimap \mathbf{N}} (n', m')$ hence by definition $n \frown_{\mathbf{N}} n'$ implies $m \frown_{\mathbf{N}} m'$ and $n \smile_{\mathbf{N}} n'$ implies $m \smile_{\mathbf{N}} m'$. Since $a \frown_{\mathbf{N}} b$ if and only if $a = b$, then either $(n, m) = (n', m')$ or $n \neq n'$. □

Let $\lambda x^{\mathbf{N}}.n$ be the non-strict function associating the natural number n to all possible inputs (also the undefined one). We name such kind of functions *erasing functions*. The above lemma assures that the class of linear functions does not contains *erasing functions*. Moreover it shows that the set of linear function contains all the stable functions except the erasing ones.

6.3. A Semantically Linear Programming Language

In this section we introduce $\mathcal{SL}$PCF, the $\mathcal{S}$emantically-$\mathcal{L}$inear PCF language. $\mathcal{SL}$PCF is a syntactical restriction of PCF. We introduce $\mathcal{SL}$PCF as a typed language, we describe terms by stressing the syntactical constraints in their formation. We define the operational semantics through an evaluation mechanism in the style of a big-step structural operational semantics. Finally we introduce the interpretation of $\mathcal{SL}$PCF terms in linear coherence spaces.

6.3.1. $\mathcal{SL}$PCF

Definition 44. *The set $\mathbb{T}$ of* linear types *is inductively defined as:*

$$\sigma ::= \iota \mid (\sigma \multimap \sigma)$$

where ι is the only ground *type i.e. the type of natural number.*

Term	*Condition*	**Free Variables**
$\mathtt{0}^{\iota}$		$\mathrm{FV}(\mathtt{0}^{\iota}) = \emptyset$
$\mathtt{s}^{\iota \multimap \iota}$		$\mathrm{FV}(\mathtt{s}^{\iota \multimap \iota}) = \emptyset$
$\mathtt{p}^{\iota \multimap \iota}$		$\mathrm{FV}(\mathtt{p}^{\iota \multimap \iota}) = \emptyset$
$\varkappa^{\sigma}$	if $\varkappa^{\sigma} \in \mathrm{Var}^{\sigma} \cup \mathrm{SVar}^{\sigma}$	$\mathrm{FV}(\varkappa^{\sigma}) = \{\varkappa^{\sigma}\}$
$(\lambda \mathtt{x}^{\sigma}.\mathtt{M}^{\tau})^{\sigma \multimap \tau}$	if $\mathtt{x}^{\sigma} \in \mathrm{Var}^{\iota}$ or $\mathtt{x}^{\sigma} \in \mathrm{HFV}(\mathtt{M}^{\tau})$	$\mathrm{FV}((\lambda \mathtt{x}^{\sigma}.\mathtt{M}^{\tau})^{\sigma \multimap \tau})$ $= \mathrm{FV}(\mathtt{M}^{\tau}) - \{\mathtt{x}^{\sigma}\}$
$(\mathtt{M}^{\sigma \multimap \tau}\mathtt{N}^{\sigma})^{\tau}$	if $\mathrm{HFV}(\mathtt{M}^{\sigma \multimap \tau}) \cap \mathrm{HFV}(\mathtt{N}^{\sigma}) = \emptyset$	$\mathrm{FV}((\mathtt{M}^{\sigma \multimap \tau}\mathtt{N}^{\sigma})^{\tau})$ $= \mathrm{FV}(\mathtt{M}^{\sigma \multimap \tau}) \cup \mathrm{FV}(\mathtt{N}^{\sigma})$
$(\mathtt{if}\ \mathtt{M}^{\iota}\ \mathtt{L}^{\iota}\ \mathtt{R}^{\iota})^{\iota}$	if $\mathrm{HFV}(\mathtt{L}^{\iota}) = \mathrm{HFV}(\mathtt{R}^{\iota})$ and $\mathrm{HFV}(\mathtt{M}^{\iota}) \cap \mathrm{HFV}(\mathtt{R}^{\iota}) = \emptyset$	$\mathrm{FV}((\mathtt{if}\ \mathtt{M}^{\iota}\ \mathtt{L}^{\iota}\ \mathtt{R}^{\iota})^{\iota})$ $= \mathrm{FV}(\mathtt{M}^{\iota}) \cup \mathrm{FV}(\mathtt{L}^{\iota}) \cup \mathrm{FV}(\mathtt{R}^{\iota})$
$(\mu F^{\sigma}.\mathtt{M}^{\sigma})^{\sigma}$	if $F^{\sigma} \in \mathrm{SFV}(\mathtt{M}^{\sigma})$ and $\mathrm{HFV}(\mathtt{M}^{\sigma}) = \emptyset$	$\mathrm{FV}((\mu F^{\sigma}.\mathtt{M}^{\sigma})^{\sigma})$ $= \mathrm{FV}(\mathtt{M}^{\sigma}) - \{F^{\sigma}\}$

Table 6.1.: $\mathcal{S}\ell$PCF: Semantically-ℓinear PCF.

Linear types are ranged over by σ, τ, μ. It is easy to see that all types τ have the shape $\tau_1 \multimap ... \multimap \tau_n \multimap \iota$, for some type $\tau_1, ..., \tau_n$ where $n \geq 0$.
We need to distinguish between different kinds of variables. Associated with every type $\sigma \in \mathbb{T}$ there are enumerable sets $\mathrm{Var}^{\sigma}, \mathrm{SVar}^{\sigma}$ of variables of type σ. $\mathrm{HVar} = \bigcup_{\sigma,\tau \in \mathbb{T}} \mathrm{Var}^{\sigma \multimap \tau}$ is the set of *linear higher-order* variables, $\mathrm{SVar} = \bigcup_{\sigma \in \mathbb{T}} \mathrm{SVar}^{\sigma}$ is the set of *stable* variables and $\mathrm{Var} = \mathrm{Var}^{\iota} \cup \mathrm{HVar} \cup \mathrm{SVar}$ is the set of variables.
Terms of $\mathcal{S}\ell$PCF and their sets of *free variables* are mutually defined in Table 6.1. We denote $\mathrm{GFV}(\mathtt{M}^{\sigma}) = \mathrm{FV}(\mathtt{M}^{\sigma}) \cap \mathrm{Var}^{\iota}$ the set of *linear ground free variables*, $\mathrm{HFV}(\mathtt{M}^{\sigma}) = \mathrm{FV}(\mathtt{M}^{\sigma}) \cap \mathrm{HVar}(\mathtt{M}^{\sigma})$ the set of *linear higher-order free variables* and $\mathrm{SFV}(\mathtt{M}^{\sigma}) = \mathrm{FV}(\mathtt{M}^{\sigma}) \cap \mathrm{SVar}$ the set of *stable free variables*.

Letters $\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots$ range over variables in Var^{σ} while $F_0, F_1, F_2, \ldots$ range over stable variables. When useful, $\varkappa$ denotes both kinds of variables. As usual $\mathtt{M}, \mathtt{N}, \mathtt{L}, \ldots$ range over terms. Sometimes types are omitted when they are clear from the context or uninteresting. Note that given the types of all variables of a term $\mathtt{M}$, there is a unique σ such that $\mathtt{M}^{\sigma}$. Moreover, as usual, $\mathtt{M}[\mathtt{N}/\varkappa]$ denotes the capture-free substitution of all free occurrences of $\varkappa$ in $\mathtt{M}$ by $\mathtt{N}$ both for stable and for linear variables.
Some comments about the terms construction follow. There are no truth-values since they are coded on integers, as usual zero codes *true* while any other numeral stands for *false*. Linear ground variables can occurs more than once in terms and it is possible to λ-abstract also linear ground variables not occurring in the body of the λ-abstraction. Moreover, they cannot be abstracted by the μ-abstraction. Conversely, stable variables

can be abstracted only by μ-abstraction and only when they are free and there are no free higher-order variables in the body of the μ-abstraction. This to preserve the constraints on linear higher-order variables during the computation.
The higher-order variables require a more careful treatment. To better understand some of the constraints on the higher-order variables in Table 6.1, it could be useful to adapt the notion of sliced occurrences of a variable in a term, defined in Definition 34, to the case of $\mathcal{SL}$PCF. This can be done as follows.

Definition 45. *The number of* sliced occurrences *of a variable* $\mathtt{x} \in \mathrm{HVar}$ *in* $\mathtt{M}^{\sigma}$ *is inductively defined as:*

$$n_{so}(\mathtt{x}, \mathtt{0}) = n_{so}(\mathtt{x}, \mathtt{1}) = n_{so}(\mathtt{x}, \mathtt{s}) = n_{so}(\mathtt{x}, \mathtt{p}) = n_{so}(\mathtt{x}, \mathtt{y}) = n_{so}(\mathtt{x}, F) = 0$$
$$n_{so}(\mathtt{x}, \mathtt{x}) = 1 \qquad n_{so}(\mathtt{x}, \lambda \mathtt{y}.\mathtt{N}) = n_{so}(\mathtt{x}, \mathtt{N}) \qquad n_{so}(\mathtt{x}, \mathtt{PQ}) = n_{so}(\mathtt{x}, \mathtt{P}) + n_{so}(\mathtt{x}, \mathtt{Q})$$
$$n_{so}(\mathtt{x}, \mathtt{if\ N\ L\ R}) = \max\{n_{so}(\mathtt{x}, \mathtt{N}), n_{so}(\mathtt{x}, \mathtt{L}), n_{so}(\mathtt{x}, \mathtt{R})\} \qquad n_{so}(\mathtt{x}, \mu F.\mathtt{N}) = n_{so}(\mathtt{x}, \mathtt{N})$$

Now, some of the constraints on higher-order variables are necessary to ensure that such variables can be used at most once during the computation. This is expressed by the following lemma.

Lemma 89. *Let* $\mathtt{M} \in \mathcal{SL}$PCF*. Then for every* $\mathtt{x} \in \mathrm{HVar}$*:* $n_{so}(\mathtt{x}, \mathtt{M}) \leq 1$*.*

Proof. Easy, by induction on the structure of $\mathtt{M}$. The base cases are trivial, the other cases follow by induction hypothesis analyzing the constraints on the terms formation and the definition of sliced occurrences of a variable in a term. □

Higher-order variables can be λ-abstracted only if they effectively occurs free in the body of the λ-abstraction. Moreover, the constraint $\mathrm{HFV}(\mathtt{L}) = \mathrm{HFV}(\mathtt{R})$ in the construction of the term $\mathtt{if\ M\ L\ R}$ assures that, in the computation, linear higher-order variables cannot disappear by the erasure of an `if` branch. These further constraints assure the strictness for higher-order abstractions.
In the sequel, it will be useful the following characterization of the shape of $\mathcal{SL}$PCF terms.

Lemma 90. *If* $\mathtt{M} \in \mathcal{SL}$PCF *then there are unique* $n, m \geq 0$ *such that*

$$\mathtt{M} = \lambda \mathtt{x}_1 \ldots \mathtt{x}_n.\zeta \mathtt{M}_1 \ldots \mathtt{M}_m$$

for some $\mathtt{M}_1, \ldots, \mathtt{M}_m \in \mathcal{SL}$PCF*, and* ζ *is either a constant (between* $\mathtt{0}, \mathtt{s}, \mathtt{p}$*), a variable, a term of the shape* $\mathtt{if\ N\ L\ R}$ *for some* $\mathtt{N}, \mathtt{L}, \mathtt{R} \in \mathcal{SL}$PCF*, a term of the shape* $\mu F.\mathtt{N}$ *for some* $\mathtt{N} \in \mathcal{SL}$PCF *or a term of the shape* $(\lambda \mathtt{x}.\mathtt{P})\mathtt{Q}$ *for some* $\mathtt{P}, \mathtt{Q} \in \mathcal{SL}$PCF*.*

Proof. By induction on terms construction. □

In general $\mathcal{SL}\mathrm{PCF}$ terms are not closed by substitution, i.e. $\mathtt{M}^\tau, \mathtt{N}^\sigma \in \mathcal{SL}\mathrm{PCF}$ and $\varkappa^\sigma \in \mathrm{FV}(\mathtt{M}^\tau)$ do not imply $\mathtt{M}^\tau[\mathtt{N}^\sigma/\varkappa^\sigma] \in \mathcal{SL}\mathrm{PCF}$. Nevertheless we have the following lemma.

Lemma 91. *Let* $\mathtt{M}^\tau \in \mathcal{SL}\mathrm{PCF}$ *and* $\varkappa_1^{\sigma_1}, \ldots, \varkappa_n^{\sigma_n} \in \mathrm{FV}(\mathtt{M}^\tau)$. *Then for every* $\mathtt{N}_1^{\sigma_1}, \ldots, \mathtt{N}_n^{\sigma_n} \in \mathcal{SL}\mathrm{PCF}$ *such that* $\mathrm{HFV}(\mathtt{N}_1^{\sigma_1}) = \cdots = \mathrm{HFV}(\mathtt{N}_n^{\sigma_n}) = \emptyset$*:*

$$\mathtt{M}^\tau[\mathtt{N}_1^{\sigma_1}/\varkappa_1^{\sigma_1}, \ldots, \mathtt{N}_n^{\sigma_n}/\varkappa_n^{\sigma_1}] \in \mathcal{SL}\mathrm{PCF}$$

Proof. Easy, by induction on $\mathtt{M}^\tau$. The base cases are trivial, the other cases follow directly by inductive hypothesis since the terms $\mathtt{N}_i^{\sigma_i}$ satisfy every constraints of Table 6.1. □

6.3.2. Structural Operational Semantics

In what follows we define the operational semantics associated to $\mathcal{SL}\mathrm{PCF}$. The operational semantics is given through an evaluation relation in the style of big-step structural operational semantics [Plotkin, 2004, Kahn, 1987]. As usual programs are closed terms of ground type.

Definition 46. *The set of* programs $\mathcal{P}$ *is the set of terms* $\mathtt{M}^\iota \in \mathcal{SL}\mathrm{PCF}$ *such that* $\mathrm{FV}(\mathtt{M}^\iota) = \emptyset$.

We are particularly interested in programs of a particular canonical form representing natural numbers.

Definition 47. *Let* $\underline{\mathtt{n}}$ *denotes the term* $\mathtt{s}(\cdots(\mathtt{s}(\mathtt{0}))\cdots)$ *where* $\mathtt{s}$ *is applied* n*-times to* $\mathtt{0}$. *The set of* numerals $\mathcal{N}$ *is the set of terms of the shape* $\underline{\mathtt{n}}$ *for some* $n \in \mathbb{N}$.

Now we are ready to introduce the evaluation mechanism of $\mathcal{SL}\mathrm{PCF}$.

Definition 48. *The* evaluation relation $\Downarrow \subseteq \mathcal{P} \times \mathcal{N}$ *is the effective relation inductively defined by the rules of Table 6.2. If there exists a numeral* $\underline{\mathtt{n}} \in \mathcal{N}$ *such that* $\mathtt{M} \Downarrow \underline{\mathtt{n}}$ *then we say that* $\mathtt{M}$ converges*, and we write* $\mathtt{M} \Downarrow$*, otherwise we say that it* diverges*, and we write* $\mathtt{M} \Uparrow$.

Some comments follow. The characterization of lemma 90 assures the determinism of rules $(\lambda^\iota), (\lambda^{-\circ})$ and (μ). Moreover, Lemma 91 assures that the evaluation, in particular for what concern the rule $(\lambda^{-\circ})$, is well given.
In fact the relation $\Downarrow$ implements a *call-by-value* parameter passing policy in the case

$$\frac{}{\mathtt{0} \Downarrow \underline{\mathtt{0}}}\ (0) \qquad \frac{\mathtt{M} \Downarrow \underline{\mathtt{n}}}{\mathtt{s}(\mathtt{M}) \Downarrow \mathtt{s}(\underline{\mathtt{n}})}\ (s) \qquad \frac{\mathtt{M} \Downarrow \underline{\mathtt{0}}}{\mathtt{p}(\mathtt{M}) \Downarrow \underline{\mathtt{0}}}\ (p_0) \qquad \frac{\mathtt{M} \Downarrow \mathtt{s}(\underline{\mathtt{n}})}{\mathtt{p}(\mathtt{M}) \Downarrow \underline{\mathtt{n}}}\ (p_n)$$

$$\frac{\mathtt{M} \Downarrow \underline{\mathtt{0}} \quad \mathtt{L} \Downarrow \underline{\mathtt{m}}}{(\mathtt{if}\ \mathtt{M}\ \mathtt{L}\ \mathtt{R}) \Downarrow \underline{\mathtt{m}}}\ (\mathrm{if}_l) \qquad \frac{\mathtt{M} \Downarrow \mathtt{s}(\underline{\mathtt{n}}) \quad \mathtt{R} \Downarrow \underline{\mathtt{m}}}{(\mathtt{if}\ \mathtt{M}\ \mathtt{L}\ \mathtt{R}) \Downarrow \underline{\mathtt{m}}}\ (\mathrm{if}_r) \qquad \frac{\mathtt{M}[\mu F.\mathtt{M}/F]\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}{(\mu F.\mathtt{M})\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}\ (\mu)$$

$$\frac{\mathtt{N} \Downarrow \underline{\mathtt{m}} \quad \mathtt{M}[\underline{\mathtt{m}}/\mathtt{x}]\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}{(\lambda \mathtt{x}^\iota.\mathtt{M})\mathtt{N}\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}\ (\lambda^\iota) \qquad \frac{\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}{(\lambda \mathtt{x}^{\sigma \multimap \tau}.\mathtt{M})\mathtt{N}\mathtt{P}_1 \cdots \mathtt{P}_\mathtt{i} \Downarrow \underline{\mathtt{n}}}\ (\lambda^\multimap)$$

Table 6.2.: Operational Semantics of $\mathcal{SL}$PCF.

of ground abstraction and *call-by-name* parameter passing policy in the case of higher-order abstraction. Moreover it implement a *lazy* (or *weak*) evaluation strategy. Note that the evaluation of $\mathtt{if}\ \mathtt{M}^\iota\ \mathtt{L}^\iota\ \mathtt{R}^\iota$ asks to evaluate exactly one subterm between $\mathtt{L}^\iota$ and $\mathtt{R}^\iota$.
The usual notion of context can be generalized to the case of $\mathcal{SL}$PCF as follows.

Definition 49. *Let $[\sigma]$ be a* special constant *of type σ. The set of σ-*context Ctx_σ *is generated by the following grammar:*

$$C[\sigma] ::= \ [\sigma] \mid \mathtt{x}^\tau \mid F^\tau \mid \mathtt{0} \mid \mathtt{s} \mid \mathtt{p} \mid (\mathtt{if}\ C[\sigma]\ C[\sigma]\ C[\sigma]) \mid (\lambda \mathtt{x}.C[\sigma]) \mid (C[\sigma]C[\sigma]) \mid \mu F.C[\sigma]$$

$C[\mathtt{N}^\sigma]$ denotes the result obtained by replacing all the occurrences of $[\sigma]$ in the context $C[\sigma]$ by the term $\mathtt{N}^\sigma$, allowing the capture of its free variables.

Note again that in general $\mathtt{N}^\sigma \in \mathcal{SL}$PCF and $C[\sigma] \in \mathrm{Ctx}_\sigma$ do not imply that $C[\mathtt{N}^\sigma] \in \mathcal{SL}$PCF. $\mathcal{SL}$PCF can be studied by considering the usual operational preorder and equivalence.

Definition 50 (Operational Equivalence). *Let $\mathtt{M}^\sigma, \mathtt{N}^\sigma \in \mathcal{SL}$PCF.*

1. $\mathtt{M} \lessapprox_\sigma \mathtt{N}$ *whenever for each context $C[\sigma]$ such that $C[\mathtt{M}], C[\mathtt{N}] \in \mathcal{P}$, if $C[\mathtt{M}] \Downarrow \underline{\mathtt{n}}$ then $C[\mathtt{N}] \Downarrow \underline{\mathtt{n}}$.*

2. $\mathtt{M} \approx_\sigma \mathtt{N}$ *if and only if* $\mathtt{M} \lessapprox_\sigma \mathtt{N}$ *and* $\mathtt{N} \lessapprox_\sigma \mathtt{M}$.

It is easy to verify that $\lessapprox_\sigma$ is a preorder while $\approx_\sigma$ is a congruence. For the sequel, it is convenient to define some abbreviations. As usual, the symbol $\doteq$ denotes the definitional equivalence.

$$\Omega^\iota \doteq \mu F^\iota.F^\iota$$

and if $\sigma_0 = \mu_1 \multimap ... \multimap \mu_m \multimap \iota$, for some $m \in \mathbb{N}$, then:

$$\Omega^{\sigma_0 \multimap ... \multimap \sigma_n \multimap \iota} \doteq \lambda x_0^{\sigma_0} \ldots x_n^{\sigma_n}.\mathtt{if}(\Omega^{\sigma_1 \multimap ... \multimap \sigma_n \multimap \iota} x_1^{\sigma_1} \ldots x_n^{\sigma_n})(x_0 \Omega^{\mu_1} \ldots \Omega^{\mu_m})(x_0 \Omega^{\mu_1} \ldots \Omega^{\mu_m})$$

Ω^σ can be used to define the approximants of a fixpoint $\mu F.M^\sigma$.

$$\mu^0 F.M^\sigma \doteq \Omega^\sigma \qquad\qquad \mu^{n+1} F.M^\sigma \doteq M[\mu^n F.M/F]$$

The following lemma describes the operational behaviour of the above terms.

Lemma 92. *Let $M_0^{\sigma_0}, ..., M_m^{\sigma_m}$ be closed $\mathcal{SL}$PCF terms ($m \geq 0$). Then:*

1. *$\Omega^{\sigma_0 \multimap ... \multimap \sigma_m \multimap \iota} M_0 ... M_m$ is a program and $\Omega^{\sigma_0 \multimap ... \multimap \sigma_m \multimap \iota} M_0 ... M_m \Uparrow$.*
2. *Let $(\mu F.P^\sigma) M_0 ... M_m$ be a program. Then, $(\mu F.P^\sigma) M_0 ... M_m \Downarrow \underline{n}$ if and only if $(\mu^k F.P^\sigma) M_0 ... M_m \Downarrow \underline{n}$, for some $k \in \mathbb{N}$.*

Proof.

1. Clearly every $\Omega^{\sigma_0 \multimap ... \multimap \sigma_m \multimap \iota} M_0 ... M_m$ is a closed term of ground type. The proof that they diverge can be done by induction on m. The base case is easy. The other cases follows directly by induction hypothesis.
2. The proof is quite involved, but standard. See for example [Gunter, 1992], Section 4.4. □

6.3.3. Linear standard interpretation

In what follows we are concerned with a standard interpretation of $\mathcal{SL}$PCF

Definition 51. *An interpretation is* standard *when ground types are interpreted on flat partially ordered sets.*

Emphatic brackets are used as notation in order to formalize both the correspondence between types and coherence spaces and the correspondence between terms and cliques, in particular $[\![\iota]\!] = \mathbf{N}$ and $[\![\sigma \multimap \tau]\!] = [\![\sigma]\!] \multimap [\![\tau]\!]$. Let $[\![\tau_0 \multimap ... \multimap \tau_m \multimap \iota]\!]$ be a coherence space; for sake of simplicity, its tokens are written as $(a_1, ..., a_m, b)$ where $a_i \in |[\![\tau_i]\!]|$ for each $i \leq m$ and $b \in |\mathbf{N}|$.
In order to interpret open terms of $\mathcal{SL}$PCF as usual we need to introduce environments.

Definition 52. *An* environment *ρ is a function that associates to each variable x^σ a clique $x \in Cl([\![\sigma]\!])$. The set of environments is denoted by* Env. *If $x \in Cl([\![\sigma]\!])$ and $x^\sigma \in$ Var then $\rho[x := x]$ is the environment such that, $\rho[x := x](x) = x$, but if $x' \neq x$ then $\rho[x := x](x') = \rho(x')$.*

$$[\![\varkappa^\sigma]\!]\rho = \rho(\varkappa^\sigma) \qquad [\![\mathtt{0}^\iota]\!]\rho = \{0\} \qquad [\![\mathtt{s}^{\iota\multimap\iota}]\!]\rho = \{(n, n+1) \mid n \in \mathbb{N}\}$$

$$[\![\mathtt{p}^{\iota\multimap\iota}]\!]\rho = \{(0,0)\} \cup \{(n, n-1) \mid n > 0\}$$

$$[\![(\mathtt{if}\ \mathtt{M}^\iota\ \mathtt{N}^\iota\ \mathtt{L}^\iota)^\iota]\!]\rho = \begin{array}{l}\{n \in \mathbb{N} \mid [\![\mathtt{M}^\iota]\!]\rho = \{0\} \wedge [\![\mathtt{N}^\iota]\!]\rho = \{n\}\} \cup \\ \{n \in \mathbb{N} \mid [\![\mathtt{M}^\iota]\!]\rho = \{m+1\} \wedge [\![\mathtt{L}^\iota]\!]\rho = \{n\}, m \in \mathbb{N}\}\end{array}$$

$$[\![\lambda\mathtt{x}^\sigma.\mathtt{M}^\tau]\!]\rho = \{(a_0, b) \ \in [\![\sigma]\!] \times [\![\tau]\!] \mid b \in [\![\mathtt{M}^\tau]\!]\rho[\mathtt{x}^\sigma := \{a_0\}]\}$$

$$[\![\mathtt{M}^{\sigma\multimap\tau}\mathtt{N}^\sigma]\!]\rho = \mathcal{F}([\![\mathtt{M}^{\sigma\multimap\tau}]\!]\rho)[\![\mathtt{N}^\sigma]\!]\rho \qquad [\![(\mu F^\sigma.\mathtt{M}^\sigma)^\sigma]\!]\rho = \mathtt{fix}(\lambda\!\!\lambda x.[\![\mathtt{M}]\!]\rho[F := x])$$

Table 6.3.: Interpretation Map.

We are now ready to introduce the linear interpretation of $\mathcal{SL}$PCF terms.

Definition 53. *Let* $\mathtt{M}^\sigma \in \mathcal{SL}\mathrm{PCF}$ *and* $\rho \in \mathrm{Env}$. *Then, the* linear interpretation *of* $\mathtt{M}^\sigma$ *in the environment* ρ, *denoted* $[\![\mathtt{M}^\sigma]\!]\rho$ *is defined in Table 6.3.*

Note that the map $\mathcal{F}$ has been defined in Lemma 83, while `fix` has been defined in Theorem 25. Moreover, note that $\lambda\!\!\lambda$ represent the semantical function abstraction. Hence, it is easy to verify the following lemma.

Lemma 93. *Let* $\mathtt{M}^\sigma \in \mathcal{SL}\mathrm{PCF}$ *and* $\rho \in \mathrm{Env}$. *Then* $[\![\mathtt{M}^\sigma]\!]\rho \in \mathcal{C}l([\![\sigma]\!])$

As usual the interpretation induces an equivalence relation over terms of $\mathcal{SL}$PCF.

Definition 54. *Let* $\mathtt{M}^\sigma, \mathtt{N}^\sigma \in \mathcal{SL}\mathrm{PCF}$. *Then,* $\mathtt{M}^\sigma$ *and* $\mathtt{N}^\sigma$ *are* denotationally equivalent, *denoted* $\mathtt{M}^\sigma \backsim_\sigma \mathtt{N}^\sigma$, *if and only if* $[\![\mathtt{M}^\sigma]\!]\rho = [\![\mathtt{N}^\sigma]\!]\rho$ *for every* $\rho \in \mathrm{Env}$.

The interpretation of closed terms is invariant with respect to environments, thus in such cases the environment can be omitted. Some basic properties of a lambda-model are recalled in the next lemma.

Lemma 94. *Let* $\mathtt{M}^\sigma, \mathtt{N}^\tau \in \mathcal{SL}\mathrm{PCF}$ *and* $\rho, \rho' \in \mathrm{Env}$.

1. *If* $\rho(\varkappa) \subseteq \rho'(\varkappa)$ *for each* $\varkappa \in \mathrm{FV}(\mathtt{M})$, *then* $[\![\mathtt{M}]\!]\rho \subseteq [\![\mathtt{M}]\!]\rho'$.
2. *If* $\mathtt{M}^\sigma[\mathtt{N}/\varkappa^\tau] \in \mathcal{SL}\mathrm{PCF}$ *then* $[\![\mathtt{M}^\sigma[\mathtt{N}/\varkappa^\tau]]\!]\rho = [\![\mathtt{M}]\!]\rho[\varkappa^\tau := [\![\mathtt{N}]\!]\rho]$.
3. *Let* $\sigma = \tau$ *and* $C[^\sigma] \in \mathrm{Ctx}_\sigma$ *such that* $C[\mathtt{M}], C[\mathtt{N}] \in \mathcal{P}$. *If* $[\![\mathtt{M}]\!]\rho = [\![\mathtt{N}]\!]\rho$ *then* $[\![C[\mathtt{M}]]\!] = [\![C[\mathtt{N}]]\!]$.

Proof.

1. By induction on the structure of $\mathtt{M}^\sigma$. The base cases $\mathtt{M} \equiv \varkappa, \mathtt{M} \equiv \mathtt{0}, \mathtt{M} \equiv \mathtt{s}$ and $\mathtt{M} \equiv \mathtt{p}$ are obvious. The cases $\mathtt{M} \equiv \mathtt{if\ N\ L\ R}$, $\mathtt{M} \equiv \lambda \mathtt{x.N}$ and $\mu F.\mathtt{N}$ follow directly by induction hypothesis. The case $\mathtt{M} \equiv \mathtt{PQ}$ follows directly by induction hypothesis and monotonicity of the map $\mathcal{F}$.

2. By induction on the structure of $\mathtt{M}^\sigma$. The base cases $\mathtt{M} \equiv \varkappa, \mathtt{M} \equiv \mathtt{0}, \mathtt{M} \equiv \mathtt{s}$ and $\mathtt{M} \equiv \mathtt{p}$ are obvious. The other cases follow directly by induction hypothesis using the previous point of this lemma in the case $\varkappa$ does not occurs in $\mathtt{M}$.

3. By induction on the structure of $C[^\sigma]$. The base cases where $C[^\sigma]$ is either $[^\sigma], \mathtt{x}^\tau, F^\tau, \mathtt{0}, \mathtt{s}$ or $\mathtt{p}$ are obvious. The other cases follow directly by induction hypothesis. □

As we expect, the interpretation of fixpoints enjoys the following property.

Lemma 95.

$$[\![\mu F.\mathtt{M}^\sigma]\!]\rho = [\![\mathtt{M}^\sigma]\!]\rho[F := [\![\mu F.\mathtt{M}^\sigma]\!]\rho]$$

Proof. By Theorem 25 $f(\mathtt{fix}(f)) = \mathtt{fix}(f)$, so in particular :

$$\begin{aligned}
[\![(\mu F^\sigma.\mathtt{M}^\sigma)^\sigma]\!]\rho &= \mathtt{fix}(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x]) \\
&= (\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])\mathtt{fix}(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x]) \\
&= [\![\mathtt{M}]\!]\rho[F := \mathtt{fix}(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])] \\
&= [\![\mathtt{M}^\sigma]\!]\rho[F := [\![\mu F.\mathtt{M}^\sigma]\!]\rho]
\end{aligned}$$

□

The following lemma shows that the linear interpretation is well defined with respect to the approximants of the fixpoint operator.

Lemma 96. $[\![\mu F.\mathtt{M}^\sigma]\!]\rho = \bigcup_{n\in\mathbb{N}} [\![\mu^n F.\mathtt{M}^\sigma]\!]\rho$, *for each* $\sigma \in \mathbb{T}$.

Proof. By Theorem 25 $\mathtt{fix}(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x]) = \bigcup_{n\in\mathbb{N}} (\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])^n\emptyset$. Hence we can prove by induction on n that $(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])^n\emptyset = [\![\mu^n F.\mathtt{M}^\sigma]\!]\rho$.
The base case is easy, in fact $[\![\mu^0 F.\mathtt{M}^\sigma]\!]\rho = \emptyset$. For the inductive step, by definition:

$$(\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])^{n+1}\emptyset = [\![\mathtt{M}]\!]\rho[F := (\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])^n\emptyset]$$

and by induction hypothesis:

$$[\![\mathtt{M}]\!]\rho[F := (\mathbb{\lambda} x.[\![\mathtt{M}]\!]\rho[F := x])^n\emptyset] = [\![\mathtt{M}]\!]\rho[F := [\![\mu^n F.\mathtt{M}^\sigma]\!]\rho]$$

finally, since by Lemma 94.2 $[\![\mathtt{M}^\sigma]\!]\rho[F := [\![\mu^n F.\mathtt{M}^\sigma]\!]\rho] = [\![\mu^{n+1} F.\mathtt{M}^\sigma]\!]\rho$, the conclusion follows. □

In what follows we are particularly interested in the relations between the operational and the denotational semantics.

Definition 55. *Let* $\mathtt{M}^\sigma, \mathtt{N}^\sigma \in \mathcal{SL}\mathrm{PCF}$. *Then:*

- *the linear interpretation is* correct *with respect to the operational semantics if* $\mathtt{M} \backsim_\sigma \mathtt{N}$ *implies* $\mathtt{M} \approx_\sigma \mathtt{N}$.
- *the linear interpretation is* complete *with respect to the operational semantics if* $\mathtt{M} \approx_\sigma \mathtt{N}$ *implies* $\mathtt{M} \backsim_\sigma \mathtt{N}$.
- *the linear interpretation is* fully abstract *with respect to the operational semantics if* $\mathtt{M} \approx_\sigma \mathtt{N}$ *if and only if* $\mathtt{M} \backsim_\sigma \mathtt{N}$.

6.4. Correctness of Interpretation

In this section we show that the linear interpretation presented in Definition 53 is correct with respect to the operational semantics presented in Definition 48. To show this we firstly prove, by means of the computability technique, that the linear interpretation is adequate for the operational semantics.

6.4.1. Computability

In order to show that our interpretation is adequate we straightforward adapt the proof of Plotkin [Plotkin, 1977] for Scott-continuous domains, based on a computability argument in Tait style.

Definition 56. *The* computability predicate *is defined as:*

- *Case* $\mathrm{FV}(\mathtt{M}^\sigma) = \emptyset$.
  - *Subcase* $\sigma = \iota$. $\mathrm{Comp}(\mathtt{M}^\iota)$ *if and only if* $[\![\mathtt{M}]\!]\rho = [\![\underline{\mathtt{n}}]\!]\rho$ *implies* $\mathtt{M} \Downarrow \underline{\mathtt{n}}$.
  - *Subcase* $\sigma = \mu \multimap \tau$. $\mathrm{Comp}(\mathtt{M}^{\mu\multimap\tau})$ *if and only if* $\mathrm{Comp}(\mathtt{M}^{\mu\multimap\tau}\mathtt{N}^\mu)$ *for each closed* $\mathtt{N}^\mu$ *such that* $\mathrm{Comp}(\mathtt{N}^\mu)$.
- *Case* $\mathrm{FV}(\mathtt{M}^\sigma) = \{\varkappa_1^{\tau_1}, \ldots, \varkappa_n^{\tau_n}\}$, *for some* $n \geq 1$.
  $\mathrm{Comp}(\mathtt{M}^\sigma)$ *if and only if* $\mathrm{Comp}(\mathtt{M}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_n/\varkappa_n])$ *for each closed* $\mathtt{N}_i^{\tau_i}$ *such that* $\mathrm{Comp}(\mathtt{N}_i^{\tau_i})$.

The following lemma gives an equivalent formulation of the computability predicate.

Lemma 97. *Let* $\mathtt{M}^{\tau_1 \multimap \ldots \multimap \tau_m \multimap \iota} \in \mathcal{SL}\mathrm{PCF}$ *and* $\mathrm{FV}(\mathtt{M}) = \{\varkappa_1^{\mu_1}, ..., \varkappa_n^{\mu_n}\}$ $(n, m \in \mathbb{N})$ $\mathrm{Comp}(\mathtt{M})$ *if and only if* $[\![\mathtt{M}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_n/\varkappa_n]\mathtt{P}_1 \ldots \mathtt{P}_m]\!]\rho = [\![\underline{\mathtt{n}}]\!]$ *implies* $\mathtt{M}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_n/\varkappa_n]\mathtt{P}_1 \ldots \mathtt{P}_m \Downarrow \underline{\mathtt{n}}$ *for each closed* $\mathtt{N}_i^{\mu_i}$ *and* $\mathtt{P}_j^{\tau_j}$ *such that* $\mathrm{Comp}(\mathtt{N}_i)$ *and* $\mathrm{Comp}(\mathtt{P}_j)$ *where* $i \leq n, j \leq m$.

Proof. Easy, by inspection of the cases in the definition of computability predicate. □

We can now prove that the computability predicate holds for every $\mathcal{SL}\mathrm{PCF}$ term.

Lemma 98. *If* $\mathtt{M}^\sigma \in \mathcal{SL}\mathrm{PCF}$ *then* $\mathrm{Comp}(\mathtt{M}^\sigma)$.

Proof. The proof is done by induction on the "untyped syntax shape" of terms. At each induction step we prove that the predicate holds for each typed term with such an "untyped syntax shape".

- $\mathtt{M} \equiv \mathtt{0}$. The only possible type is ι, so the proof is obvious.
- $\mathtt{M} \equiv \varkappa$. Let $\sigma = \tau_1 \multimap \cdots \multimap \tau_m \multimap \iota$, where $m \in \mathbb{N}$. Let $\mathtt{P}^\sigma$ and $\mathtt{N}_i^{\tau_i}$ for $1 \leq i \leq m$ be closed terms such that $\mathrm{Comp}(\mathtt{P}^\sigma)$ and $\mathrm{Comp}(\mathtt{N}_i^{\tau_i})$. Then if $[\![\mathtt{PN}_1 \cdots \mathtt{N}_m]\!]\rho = [\![\underline{\mathtt{n}}]\!]\rho$ then $\mathtt{PN}_1 \cdots \mathtt{N}_m \Downarrow \underline{\mathtt{n}}$.
- $\mathtt{M} \equiv \mathtt{s}$. Clearly, $\iota \multimap \iota$ is the only possible type. Let $\mathtt{P}^\iota$ be a closed term such that $\mathrm{Comp}(\mathtt{P}^\iota)$, i.e $[\![\mathtt{P}^\iota]\!]\rho = [\![\underline{\mathtt{n}}]\!]\rho$ implies $\mathtt{P}^\iota \Downarrow \underline{\mathtt{n}}$. Assume $[\![\mathtt{s}(\mathtt{P}^\iota)]\!]\rho = [\![\underline{\mathtt{m}}]\!]\rho$. Since $[\![\mathtt{s}(\mathtt{P}^\iota)]\!]\rho = \{n + 1 \mid [\![\mathtt{P}]\!]\rho = \{n\}\}$, by operational rules $\mathtt{s}(\mathtt{P}^\iota) \Downarrow \mathtt{s}(\underline{\mathtt{m}})$.
- $\mathtt{M} \equiv \mathtt{p}$. Similar to the previous case.
- $\mathtt{M} = \mathtt{if}\ \mathtt{N}^\iota\ \mathtt{L}^\iota\ \mathtt{R}^\iota$ where $\mathrm{FV}(\mathtt{M}) = \{\varkappa_1^{\mu_1}, \ldots, \varkappa_k^{\mu_k}\}$ for $k \geq 0$. Clearly, ι is the only possible type. Let $\mathtt{N}_1^{\mu_1}, \ldots, \mathtt{N}_k^{\mu_k}$ be closed terms such that $\mathrm{Comp}(\mathtt{N}_i)$ for $1 \leq i \leq k$ and $\mathtt{M}'$ be the term $(\mathtt{if}\ \mathtt{N}\ \mathtt{L}\ \mathtt{R})[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_k/\varkappa_k]$. Suppose $[\![\mathtt{M}']\!]\rho = [\![\underline{\mathtt{n}}]\!]\rho$, thus either $[\![\mathtt{N}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_k/\varkappa_k]]\!]\rho = [\![\mathtt{0}]\!]\rho$ or $[\![\mathtt{N}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_k/\varkappa_k]]\!]\rho = [\![\mathtt{s}(\underline{\mathtt{m}})]\!]\rho$ by definition. In the former case, $[\![\mathtt{L}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_n/\varkappa_n]]\!]\rho = [\![\underline{\mathtt{n}}]\!]\rho$, so by induction hypothesis $\mathtt{N}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_k/\varkappa_k] \Downarrow \mathtt{0}$ and $\mathtt{L}[\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_n/\varkappa_n] \Downarrow \underline{\mathtt{n}}$ and applying $(\mathtt{if}_l)$ rule, the conclusion follows. The other case is similar.
- $\mathtt{M} \equiv \mathtt{NP}$. Assume $\mathtt{N}^{\tau \multimap \sigma}$ and $\mathtt{P}^\tau$ for types σ and τ. By induction hypothesis $\mathrm{Comp}(\mathtt{N}^{\tau \multimap \sigma})$ and $\mathrm{Comp}(\mathtt{P}^\tau)$ and the conclusion follows immediately.
- $\mathtt{M} \equiv \lambda\mathtt{x}.\mathtt{Q}$. Assume $\mathtt{x}^\mu$ and $\mathtt{Q}^\tau$ for types μ and τ. Let $\mathrm{FV}(\mathtt{M}) = \{\varkappa_1^{\mu_1}, \ldots, \varkappa_k^{\mu_k}\}$ for $k \geq 0$ and $\tau = \tau_1 \multimap \cdots \multimap \tau_h \multimap \iota$, where $h \geq 0$. Let $\mathtt{N}_1^{\mu_1}, \ldots, \mathtt{N}_k^{\mu_k}, \mathtt{P}_0^\mu, \mathtt{P}_1^{\tau_1}, \ldots, \mathtt{P}_h^{\tau_h}$ be closed terms such that $\mathrm{Comp}(\mathtt{N}_i)$ and $\mathrm{Comp}(\mathtt{P}_j)$ for $1 \leq i \leq k$ and $0 \leq j \leq h$ respectively. Thus $\mathrm{Comp}(\mathtt{Q}^\tau[\mathtt{P}_0/\mathtt{x}][\mathtt{N}_1/\varkappa_1, \ldots, \mathtt{N}_k/\varkappa_k]\mathtt{P}_1 \ldots \mathtt{P}_h)$, since

$\mathrm{Comp}(\mathtt{Q}^\tau)$ holds by induction hypothesis.
Consider the case $\mu \neq \iota$ and suppose $[\![(\lambda \mathtt{x}^\mu.\mathtt{Q}^\tau)[\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k]\mathtt{P}_0\dots\mathtt{P}_h]\!]\rho = [\![\underline{\mathtt{n}}]\!]$. Thus $[\![\mathtt{Q}^\tau[\mathtt{P}_0/\mathtt{x}][\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k]\mathtt{P}_1\dots\mathtt{P}_h]\!]\rho = [\![\underline{\mathtt{n}}]\!]$ by Lemma 94. Therefore $\mathtt{Q}^\tau[\mathtt{P}_0/\mathtt{x}][\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k]\mathtt{P}_1\dots\mathtt{P}_h \Downarrow \underline{\mathtt{n}}$ by induction hypothesis. The conclusion follows by the evaluation rule $(\lambda^{-\circ})$.
Now, suppose $\mu = \iota$ and $[\![(\lambda \mathtt{x}^\mu.\mathtt{Q}^\tau)[\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k]\mathtt{P}_0\dots\mathtt{P}_h]\!]\rho = [\![\underline{\mathtt{n}}]\!]$, so by definition of $\mathcal{F}$ it is necessary that $[\![\mathtt{P}_\mathtt{0}]\!] = [\![\underline{\mathtt{m}}]\!]$ for some $\underline{\mathtt{m}}$. But $\mathrm{Comp}(\mathtt{P}_\mathtt{0}^\mu)$, thus $[\![\mathtt{P}_\mathtt{0}]\!] = [\![\underline{\mathtt{m}}]\!]$ implies $\mathtt{P}_\mathtt{0} \Downarrow \underline{\mathtt{m}}$. Hence $[\![\mathtt{Q}^\tau[\underline{\mathtt{m}}/\mathtt{x}][\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_\mathtt{k}/\varkappa_\mathtt{k}]\mathtt{P}_1\dots\mathtt{P}_\mathtt{h}]\!]\rho = [\![\underline{\mathtt{n}}]\!]$, by Lemma 94. Therefore $\mathtt{Q}^\tau[\underline{\mathtt{m}}/\mathtt{x}][\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k]\mathtt{P}_1\dots\mathtt{P}_h \Downarrow \underline{\mathtt{n}}$ by induction hypothesis. The conclusion follows by the evaluation rule (λ^ι).

- $\mathtt{M} \equiv \mu F.\mathtt{N}$. Let $\mathrm{FV}(\mathtt{M}) = \{\varkappa_1^{\mu_1},\dots,\varkappa_k^{\mu_k}\}$ for $k \geq 0$ and $\sigma = \tau_1 \multimap \cdots \multimap \tau_h \multimap \iota$, where $h \geq 0$. By induction on h.
  The case $h = 0$ is trivial, so assume $h \geq 1$. Assume $\mathtt{N}_1^{\mu_1},\dots,\mathtt{N}_k^{\mu_k}$ and $\mathtt{P}_1^{\tau_1},\dots,\mathtt{P}_h^{\tau_h}$ be closed terms such that $\mathrm{Comp}(\mathtt{N}_i)$ and $\mathrm{Comp}(\mathtt{P}_j)$ for $1 \leq i \leq k$ and $1 \leq j \leq h$ respectively. Let $[\![(\mu F.\mathtt{N}^\sigma[\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k])\mathtt{P}_1\dots\mathtt{P}_h]\!]\rho = [\![\underline{\mathtt{n}}]\!]$.
  $[\![(\mu F.\mathtt{N}^\sigma[\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k])\mathtt{P}_1\dots\mathtt{P}_h]\!]\rho = [\![(\mu^k F.\mathtt{N}^\sigma[\mathtt{N}_1/\varkappa_1,\dots,\mathtt{N}_k/\varkappa_k])\mathtt{P}_1\dots\mathtt{P}_h]\!]\rho$ for some $k \in \mathbb{N}$, by Lemma 96. Thus, by the previous points of this lemma, $\mu^k F.\mathtt{N}^\sigma[\mathtt{Q}'/\mathtt{v}_1,\dots,\mathtt{Q}'/\mathtt{v}_m]\mathtt{P}_1\dots\mathtt{P}_h \Downarrow \underline{\mathtt{n}}$. The conclusion follows by Lemma 92. □

6.4.2. Adequacy and Correctness

Now we are ready to prove the correctness of our interpretation. Firstly we show that it is adequate.

Definition 57. *An interpretation function* $[\![\ ]\!]$ *is* adequate *with respect to an operational evaluation* $\Downarrow$ *if for every program* $\mathtt{M} \in \mathcal{P}$ *and numeral* $\underline{\mathtt{n}} \in \mathcal{N}$

$$[\![\mathtt{M}]\!] = [\![\underline{\mathtt{n}}]\!] \iff \mathtt{M} \Downarrow \underline{\mathtt{n}}$$

In order to prove adequacy a further result is necessary.

Lemma 99. *Let* $\mathtt{M} \in \mathcal{P}$. *If* $\mathtt{M} \Downarrow \underline{\mathtt{n}}$ *then* $[\![\mathtt{M}]\!] = [\![\underline{\mathtt{n}}]\!]$.

Proof. By induction on the derivation proving $\mathtt{M} \Downarrow \underline{\mathtt{n}}$. The base case where the last applied rule is $(\mathtt{0})$ is obvious. In the case the last applied rule is either $(\mathtt{s}), (\mathtt{p}_\mathtt{0}), (\mathtt{p}_\mathtt{n}), (\mathtt{if}_1)$ or $(\mathtt{if}_\mathtt{r})$, then the conclusion follows immediately by induction hypothesis and definition of interpretation.
In the case the last applied rule is (λ^ι): $\mathtt{M} \equiv (\lambda \mathtt{x}.\mathtt{M}')\mathtt{N}\mathtt{P}_1\cdots\mathtt{P}_i$. By induction hypothesis $[\![\mathtt{N}]\!] = [\![\underline{\mathtt{m}}]\!]$ and $[\![\mathtt{M}'[\underline{\mathtt{m}}/\mathtt{x}]\mathtt{P}_1\cdots\mathtt{P}_i]\!] = [\![\underline{\mathtt{n}}]\!]$. Hence by Lemma 94.2 and by definition of linear

interpretation the conclusion follows
In the case the last applied rule is $(\lambda^{-\circ})$: $\mathtt{M} \equiv (\lambda \mathtt{x}.\mathtt{M}')\mathtt{N}\mathtt{P}_1 \cdots \mathtt{P}_i$. By induction hypothesis $[\![\mathtt{M}'[\mathtt{N}/\mathtt{x}]\mathtt{P}_1 \cdots \mathtt{P}_i]\!] = [\![\underline{\mathtt{n}}]\!]$. Hence again by Lemma 94.2 and by definition of linear interpretation the conclusion follows.
In the case the last applied rule is (μ): $\mathtt{M} \equiv (\mu F.\mathtt{M}')\mathtt{P}_1 \cdots \mathtt{P}_i$. By induction hypothesis $[\![\mathtt{M}'[\mu F.\mathtt{M}'/F]\mathtt{P}_1 \cdots \mathtt{P}_i]\!] = [\![\underline{\mathtt{n}}]\!]$. Hence by Lemma 94.2 and Lemma 95 the conclusion follows. □

The fact that the computability predicate holds for $\mathcal{SL}$PCF terms is essential in proving the following theorem.

Theorem 26. *The linear interpretation is adequate for* $\mathcal{SL}$PCF*.*

Proof. Lemma 98 and Definition 56 assure that $[\![\mathtt{M}]\!] = [\![\underline{\mathtt{n}}]\!]$ implies $\mathtt{M} \Downarrow \underline{\mathtt{n}}$ for any program $\mathtt{M}$ and numeral $\underline{\mathtt{n}}$. Conversely Lemma 99 assures that $\mathtt{M} \Downarrow \underline{\mathtt{n}}$ implies $[\![\mathtt{M}]\!] = [\![\underline{\mathtt{n}}]\!]$ for any program $\mathtt{M}$ and numeral $\underline{\mathtt{n}}$. □

Moreover as usual the adequacy implies the correctness.

Theorem 27. *The linear interpretation is correct for* $\mathcal{SL}$PCF*.*

Proof. Let $\mathtt{M}^\sigma$ and $\mathtt{N}^\sigma$ such that $[\![\mathtt{M}]\!]_\rho = [\![\mathtt{N}]\!]_\rho$, for each environment $\rho \in \mathrm{Env}$. Let $C[^\sigma] \in \mathrm{Ctx}_\sigma$ such that $C[\mathtt{M}], C[\mathtt{N}] \in \mathcal{P}$. If $C[\mathtt{M}] \Downarrow \underline{\mathtt{n}}$ for some value $\underline{\mathtt{n}}$, then $[\![C[\mathtt{M}]]\!] = [\![\underline{\mathtt{n}}]\!]$ by Lemma 99. Since $[\![C[\mathtt{N}]]\!] = [\![C[\mathtt{M}]]\!] = [\![\underline{\mathtt{n}}]\!]$ by Lemma 94.3, $C[\mathtt{N}] \Downarrow \underline{\mathtt{n}}$ by adequacy. By definition of operational equivalence the proof is done. □

Note that since linear functions are all stricts, see Lemma 79, the correctness implies that also our terms are strict in all arguments, for all orders.

6.5. Recursive Completeness

In this section we show that $\mathcal{SL}$PCF is recursively complete. We show that all the partial recursive functions can be programmed in it.

Definition 58. *A numeric function* $f : \mathbb{N}^m \to \mathbb{N}$ *is* $\mathcal{SL}$PCF*-programmable if there is a closed* $\mathtt{F} \in \mathcal{SL}\mathrm{PCF}$ *such that* $\mathtt{F}\underline{\mathtt{n}}_1 \ldots \underline{\mathtt{n}}_m \in \mathcal{P}$ *and:*

$$\mathtt{F}\underline{\mathtt{n}}_1 \ldots \underline{\mathtt{n}}_m \Downarrow \underline{f(n_1, \ldots, n_m)}$$

for all $n_1, \ldots, n_m \in \mathbb{N}$*. We say that* F $\mathcal{SL}$PCF*-programs* f*.*

We now recall the definition of the class of partial recursive functions. More details can be found for example in [Odifreddi, 1999].

Definition 59. *The class of* partial recursive functions *is the smallest class of functions containing the* initial functions $U_i^m(n_1,\ldots,n_m) = n_i$ *for all* $1 \leq i \leq m$, $S(n) = n+1$, $Z(n) = 0$ *and closed under* composition, primitive recursion, *and* minimalization*:*

- *if* $g : \mathbb{N}^k \to \mathbb{N}$ *and* $h_1,\ldots,h_k : \mathbb{N}^m \to \mathbb{N}$ *are partial recursive functions, then so is* $g(h_1(n_1,\ldots,n_m),\ldots,h_k(n_1,\ldots,n_m))$*;*
- *if* $g : \mathbb{N}^m \to \mathbb{N}$ *and* $h : \mathbb{N}^{m+2} \to \mathbb{N}$ *are partial recursive functions, then so is* $f : \mathbb{N}^{m+1} \to \mathbb{N}$ *defined by*

$$f(0,n_1,\ldots,n_m) = g(n_1,\ldots,n_m),$$
$$f(n+1,n_1,\ldots,n_m) = h(f(n,n_1,\ldots,n_m),n,n_1,\ldots,n_m);$$

- *if* $g : \mathbb{N}^{m+1} \to \mathbb{N}$ *is a partial recursive function, then so is* $\mu n.g(n,n_1,\ldots,n_m)$*, where* $\mu n.g(n,n_1,\ldots,n_m)$ *denotes the smallest* $n \in \mathbb{N}$*, if it exists, such that the equation* $g(n,n_1,\ldots,n_m) = 0$ *is satisfied and* $\forall k \leq n$, $g(k,n_1,\ldots,n_m)$ *is defined.*

We can now prove that all partial recursive functions are $\mathcal{SL}$PCF-programmable.

Theorem 28. *Let* f *be a partial recursive function. Then, there exists* $\mathtt{F}$ *that* $\mathcal{SL}$PCF*-programs* f*.*

Proof. Clearly S is programmable by $\mathtt{s}$ while Z is programmable by $\lambda x^\iota.\mathtt{0}$. The projection functions $\mathtt{U}_\mathtt{i}^\mathtt{m}$ are programmable by terms of the shape: $\lambda x_1^\iota \ldots \lambda x_m^\iota.x_i$.
The compositions between linear functions defined by $\mathtt{G},\mathtt{H}_1,\ldots\mathtt{H}_\mathtt{n}$. is $\mathcal{SL}$PCF-programmable by terms of the shape:

$$\lambda x_1^\iota \ldots \lambda x_m^\iota.\mathtt{G}(\mathtt{H}_1 x_1^\iota \ldots x_m^\iota)\ldots(\mathtt{H}_\mathtt{n} x_1^\iota \ldots x_m^\iota)$$

Primitive recursion on linear functions defined by $\mathtt{G},\mathtt{H}$ is $\mathcal{SL}$PCF-programmable by terms of the shape:

$$\mu F.\lambda z^\iota.\lambda x_1^\iota \ldots \lambda x_m^\iota.\ \mathtt{if}\ z^\iota\ \mathtt{then}\ (\mathtt{G}x_1^\iota \ldots x_m^\iota)\ \mathtt{else}\ (\mathtt{H}\,(F\,(\mathtt{p}\,z^\iota)x_1^\iota \ldots x_m^\iota)(\mathtt{p}\,z^\iota)x_1^\iota \ldots x_m^\iota)$$

Finally, in order to show that minimalization schema is $\mathcal{SL}$PCF-programmable, let $\mathtt{G}$ represent a recursive partial function $g : \mathbb{N}^{m+1} \to \mathbb{N}$ and let $\mathtt{F}$ be

$$(\mu F.\lambda z^\iota.\lambda x_1^\iota \ldots \lambda x_m^\iota.\ \mathtt{if}\ (\mathtt{G}z^\iota x_1^\iota \ldots x_m^\iota)\ \mathtt{then}\ z^\iota\ \mathtt{else}\ (F\,(\mathtt{s}\,z^\iota)x_1^\iota \ldots x_m^\iota)\ )\mathtt{0}.$$

Assume $\underline{n}_1,\ldots,\underline{n}_m \in \mathcal{N}$ then $\mathtt{F}\underline{n}_1 \ldots \underline{n}_m \Downarrow \underline{k}$ if and only if $\mathtt{G}\underline{k}\ \underline{n}_1 \ldots \underline{n}_m \Downarrow \mathtt{0}$ and for each $h \leq k \in \mathbb{N}$ exists $\underline{n} \in \mathcal{N}$ such that $\mathtt{G}\underline{h}\ \underline{n}_1 \ldots \underline{n}_m \Downarrow \mathtt{s}(\underline{n})$. □

6.6. Lack of Full Abstraction

In order to prove the full abstraction, essentially, we need to prove the definability of all finite cliques in our spaces, similarly to what has been done in [Plotkin, 1977, Paolini, 2006]. If x_0 is a finite clique (in a linear coherence space interpretation of a type σ) then we seek a closed term $\mathtt{M}^\sigma$ of $\mathcal{SL}$PCF such that $[\![\mathtt{M}]\!] = x_0$.
As we have already stressed the linear interpretation of $\mathcal{SL}$PCF is not fully abstract since it is not complete with respect to the operational semantics. Here we analyze in more depth why the completeness fails.
Firstly, let $\mathtt{M}, \mathtt{N} \in \mathcal{SL}$PCF. Then $\mathtt{M} := \mathtt{N}$ denotes the application to $\mathtt{M}$ and $\mathtt{N}$ of the term:

$$\mu F^{\iota\multimap\iota\multimap\iota}.\lambda x^\iota.\lambda y^\iota.\mathtt{if}\ x\ (\mathtt{if}\ y\ \underline{0}\ \underline{1})\ (\mathtt{if}\ y\ \underline{1}\ (F(px)(py)))$$

It is easy to check that

$$[\![\mathtt{M} := \mathtt{N}]\!] = \begin{cases} 0 & [\![\mathtt{M}]\!] = m = [\![\mathtt{N}]\!], \\ 1 & [\![\mathtt{M}]\!] = m \neq n = [\![\mathtt{N}]\!], \\ \emptyset & \text{otherwise.} \end{cases}$$

Consider now the clique:

$$\{(((1,3),5),7)\ ,\ (((2,4),6),8)\} \in [\![((\iota\multimap\iota)\multimap\iota)\multimap\iota]\!] \tag{6.1}$$

we want a term $\mathtt{M}$ defining it. Let $\mathtt{M}_2$ be the term $\lambda x^\iota.\mathtt{if}(x := \underline{1})\underline{3}(\mathtt{if}(x := \underline{2})\underline{4}\Omega_\iota)$. Consider the following term $\mathtt{N}$:

$$\lambda F^{((\iota\multimap\iota)\multimap\iota)}.(\lambda z.\mathtt{if}(z := \underline{5})\ \underline{7}\ (\mathtt{if}(z := \underline{6})\ \underline{8}\ \Omega))(F(\mathtt{M}_2))$$

Since $[\![\mathtt{M}_2]\!] = \{(1,3)\ ,\ (2,4)\}$, we can think that the term $\mathtt{N}$ is interpreted on the clique of Equation 6.1. In fact this is not the case, since it is interpreted on the clique:

$$\{(((1,3),5),7)\ ,\ (((1,3),6),8)\ ,\ (((2,4),5),7)\ ,\ (((2,4),6),8)\} \in [\![((\iota\multimap\iota)\multimap\iota)\multimap\iota]\!]$$

So we need to make more distinctions between tokens. Now let $\mathtt{M}_0$ and $\mathtt{M}_1$ be the terms $\lambda x^\iota.\mathtt{if}(x := \underline{1})\,\underline{3}\,\Omega_\iota$ and $\lambda x^\iota.\mathtt{if}(x := \underline{2})\,\underline{4}\,\Omega_\iota$ respectively. Consider the following term $\mathtt{N}'$:

$$\lambda F^{((\iota\multimap\iota)\multimap\iota)}.\mathtt{if}\Big(F(\mathtt{M}_2) := \underline{5}\Big)\Big(\mathtt{if}\ (F(\mathtt{M}_0) := \underline{5})\ \underline{7}\ \Omega\Big)\Big(\mathtt{if}\ (F(\mathtt{M}_1) := \underline{6})\ \underline{8}\ \Omega\Big).$$

Clearly $[\![\mathtt{M}_0]\!] = \{(1,3)\}$ and $[\![\mathtt{M}_1]\!] = \{(2,4)\}$. So we can think that the term $\mathtt{N}'$ is this time really interpreted on the clique of Equation 6.1. But unfortunately, it does not belong to $\mathcal{SL}$PCF since, constraints on the higher-order variables of the first $\mathtt{if}$ are not respected. On the other hand, the expected interpretation of the above term is straightforwardly

linear.
This problem can be overcome by extending $\mathcal{SL}$PCF with a new operator $\mathtt{which?}^{((\iota\multimap\iota)\multimap\iota)\multimap(\iota\otimes\iota)}$and with constants dealing with tensor pairs. The operational behaviour of which? is sketched in the following not constructive rule:

$$\frac{\mathtt{M}(\lambda\mathtt{x}^{\iota}.\mathtt{x}) \Downarrow \underline{\mathtt{n}} \quad \exists \underline{\mathtt{k}} : \mathtt{M}(\lambda\mathtt{x}^{\iota}.\mathtt{if}(\mathtt{x} \coloneqq \underline{\mathtt{k}})\ \underline{\mathtt{k}}\ \Omega^{\iota}) \Downarrow \underline{\mathtt{n}}}{\mathtt{which?}(\mathtt{M}^{(\iota\multimap\iota)\multimap\iota}) \Downarrow \langle \underline{\mathtt{n}}, \underline{\mathtt{k}} \rangle}$$

Note that if $[\![\mathtt{M}]\!]$ is a linear function, $\mathtt{M}(\lambda\mathtt{x}^{\iota}.\mathtt{x}) \Downarrow \underline{\mathtt{n}}$ implies that there is $\underline{\mathtt{k}}$ such that $\mathtt{M}(\lambda\mathtt{x}^{\iota}.\mathtt{if}(\mathtt{x} \coloneqq \underline{\mathtt{k}})\ \underline{\mathtt{k}}\ \Omega^{\iota}) \Downarrow \underline{\mathtt{n}}$. Essentially, the operator which?, given M returns a tensor-pair of both the output of the evaluation of M applied to the identity function and the information about the trace used by M. This kind of operators is not strange, in fact it can be programmed in both stable and strongly stable domains, as shown in [Longley, 2002, Paolini, 2006]. Moreover, we hope that it is possible to provide which? with a constructive operational evaluation. So the add of the which? operator to $\mathcal{SL}$PCF and the extension of the operational evaluation are initial steps toward the full abstraction.

Bibliography

[Abiteboul and Vianu, 1989] Abiteboul, S. and Vianu, V. (1989). Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science* (LICS '89), pages 71–79, Washington, D.C. IEEE Computer Society.

[Alves et al., 2006] Alves, S., Fernández, M., Florido, M., and Mackie, I. (2006). The power of linear functions. In Ésik, Z., editor, *Proceedings of the 20th International Workshop on Computer Science Logic* (CSL '06), volume 4207 of *Lecture Notes in Computer Science*, pages 119–134.

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

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

[Atassi et al., 2006] Atassi, V., Baillot, P., and Terui, K. (2006). Verification of ptime reducibility for system F terms via dual light affine logic. In Ésik, Z., editor, *Proceedings of the 20th International Workshop on Computer Science Logic* (CSL '06), volume 4207 of *Lecture Notes in Computer Science*, pages 150–166. Springer.

[Baillot, 2002] Baillot, P. (2002). Checking polynomial time complexity with types. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 370–382.

[Baillot, 2004] Baillot, P. (2004). Type inference for light affine logic via constraints on words. *Theoretical Computer Science*, 328(3):289–323.

[Baillot et al., 2007] Baillot, P., Coppola, P., and Lago, U. D. (2007). Light logics and optimal reduction: Completeness and complexity. In *Proceedings of the Twenty-*

Second Annual IEEE Symposium on Logic in Computer Science (LICS '07), pages 421–430. IEEE Computer Society.

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

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

[Baillot and Terui, 2005] Baillot, P. and Terui, K. (2005). A feasible algorithm for typing in elementary affine logic. In *Proceedings of the 8th International Conference on Typed Lambda-Calculi and Applications* (TLCA '05), pages 55–70.

[Barendregt and Ghilezan, 2000] Barendregt, H. and Ghilezan, S. (2000). Lambda terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(1):121–134.

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

[Bellantoni, 1992] Bellantoni, S. (1992). *Predicative recursion and computational complexity*. PhD thesis, University of Toronto.

[Bellantoni and Cook, 1992] Bellantoni, S. and Cook, S. (1992). A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2(2):97–110.

[Bellantoni et al., 2000] Bellantoni, S., Niggl, K. H., and Schwichtenberg, H. (2000). Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logics*, 104:17–30.

[Benton et al., 1993] Benton, N., Bierman, G., de Paiva, V., and Hyland, M. (1993). A term calculus for intuitionistic linear logic. In Bezem, M. and Groote, J. F., editors, *Proceedings of the 1st International Conference on Typed Lambda-Calculi and Applications* (TLCA '93), volume 664 of *Lecture Notes in Computer Science*, pages 75–90. Springer-Verlag, Berlin.

[Berry, 1978] Berry, G. (1978). Stable models of typed λ-calculi. In *Fifth International Colloquium on Automata, Languages and Programming - ICALP'78, Udine, Italy, July 17-21, 1978*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89.

[Bierman et al., 2000] Bierman, G. M., Pitts, A. M., and Russo, C. V. (2000). Operational properties of lily, a polymorphic linear lambda calculus with recursion. *Electronic Notes in Theoretical Computer Science*, 41(3).

[Bonfante et al., 2006] Bonfante, G., Kahle, R., Marion, J.-Y., and Oitavem, I. (2006). Towards an implicit characterization of NC^k. In Springer, editor, *Proceedings of the 20th International Workshop on Computer Science Logic* (CSL '06), volume 4207 of *Lecture Notes in Computer Science*, pages 212–224. Springer.

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

[Clote, 1995] Clote, P. (1995). Computational models and function algebras. In Leivant, D., editor, *Proceedings of the International Workshop on Logic and Computational Complexity (LCC'94)*, volume 960 of *Lecture Notes in Computer Science*, pages 98–130, Berlin, GER. Springer.

[Cobham, 1964] Cobham, A. (1964). The intrinsic computational difficulty of functions. In Bar-Hillel, Y., editor, *Proceedings of the International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam.

[Coppo et al., 1980] Coppo, M., Dezani-Ciancaglini, M., and Venneri, B. (1980). Principal type schemes and lambda-calculus semantics. In Seldin, J. P. and Hindley, J. R., editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 535–560. Academic Press, Inc., New York, NY.

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

[Coppola and Martini, 2001] Coppola, P. and Martini, S. (2001). Typing lambda terms in elementary logic with linear constraints. In *Proceedings of the 6th International Conference on Typed Lambda-Calculi and Applications* (TLCA '01), pages 76–90.

[Coppola and Martini, 2006] Coppola, P. and Martini, S. (2006). Optimizing optimal reduction. a type inference algorithm for elementary affine logic. *ACM Transactions on Computational Logic*, 7:219–260.

[Coppola and Ronchi della Rocca, 2003] Coppola, P. and Ronchi della Rocca, S. (2003). Principal typing in elementary affine logic. In *Proceedings of the 7th International Conference on Typed Lambda-Calculi and Applications* (TLCA '03), pages 90–104.

[Curien, 2007] Curien, P.-L. (2007). Definability and full abstraction. *Electronic Notes in Theoretical Computer Science*, 172:301–310.

[Curry, 1969] Curry, H. B. (1969). Modified functionality in combinatory logic. *Dialectica*, 21.

[Damas and Milner, 1982] Damas, L. and Milner, R. (1982). Principal type schemes for functional programs. In *Proceedings of the 9th ACM Symposium on Principles of Programming Languages*, pages 207–212, Albuquerque.

[Danos and Joinet, 2003] Danos, V. and Joinet, J.-B. (2003). Linear logic and elementary time. *Information and Computation*, 183(1):123–137.

[Danos et al., 1997] Danos, V., Joinet, J.-B., and Schellinx, H. (1997). A new deconstructive logic: Linear logic. *Journal of Symbolic Logic*, 62(3):755–807.

[de Carvalho, 2007] de Carvalho, D. (2007). *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Université Aix-Marseille 2.

[Gaboardi et al., 2008] Gaboardi, M., Marion, J.-Y., and Ronchi Della Rocca, S. (2008). 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*. to appear.

[Gaboardi and Paolini, 2007] Gaboardi, M. and Paolini, L. (2007). Syntactical, operational and denotational linearity. In *Workshop on Linear Logic, Ludics, Implicit Complexity and Operator Algebras. Dedicated to Jean-Yves Girard on his 60th birthday*, Certosa di Pontignano, Siena.

[Gaboardi and Ronchi Della Rocca, 2006] Gaboardi, M. and Ronchi Della Rocca, S. (2006). Soft linear logic and λ-calculus. In *4th International Workshop on Proof, Computation, Complexity PCC'06 . Ilmenau, Germany, July 24-25 , 2006, Proceedings*.

[Gaboardi and Ronchi Della Rocca, 2007] Gaboardi, M. and Ronchi Della Rocca, S. (2007). A soft type assignment system for λ-calculus. In Duparc, J. and Henzinger, T. A., editors, *Proceedings of the 21st International Workshop on Computer Science Logic* (CSL '07), volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer.

[Giannini and Ronchi Della Rocca, 1994] Giannini, P. and Ronchi Della Rocca, S. (1994). A type inference algorithm for a stratified polymorphic type discipline. *Information and Computation*, 109(1/2):115–173.

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

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

[Girard et al., 1992] Girard, J. Y., Scedrov, A., and Scott, P. J. (1992). Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66.

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

[Grädel et al., 2007] Grädel, E., Kolaitis, P., Libkin, L., Marx, M., Spencer, J., Vardi, M., Venema, Y., and Weinstein, S. (2007). *Finite Model Theory and its applications*. Springer.

[Gunter, 1992] Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. foundations of Computing Series. MIT Press, Cambridge, MA.

[Hindley, 1969] Hindley, J. R. (1969). The principal type-scheme of an object in combinatory logic. *Transactions of the AMS*, 146:29–60.

[Hofmann, 2000] Hofmann, M. (2000). Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 104(1-3):113–166.

[Hofmann, 2003] Hofmann, M. (2003). Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183(1):57–85.

[Hyland and Ong, 2000] Hyland, J. M. E. and Ong, C.-H. L. (2000). On full abstraction for pcf: I, II, and III. *Information and Computation*, 163(2):285–408. Preliminary draft circulated since 1994.

[Immerman, 1999] Immerman, N. (1999). *Descriptive Complexity*. Springer-Verlag, New York.

[Jones, 2001] Jones, N. (2001). The expressive power of higher-order types or, life without cons. *Journal of Functional Programming*, 11(1):55–94.

[Kahn, 1987] Kahn, G. (1987). Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag.

[Kfoury and Wells, 1994] Kfoury, A. J. and Wells, J. B. (1994). A direct algorithm for type inference in the rank-2 fragment of the second-order lambda-calculus. In *LISP and Functional Programming*, pages 196–207.

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

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

[Leivant, 1991] Leivant, D. (1991). A foundational delineation of computational feasibility. In Meyer, A. R., editor, *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science* LICS '91, pages 2–11. IEEE Computer Society.

[Leivant and Marion, 1993] Leivant, D. and Marion, J.-Y. (1993). Lambda calculus characterizations of poly-time. In *Proceedings of the 1st International Conference on Typed Lambda-Calculi and Applications* (TLCA '93), volume 664 of *Lecture Notes in Computer Science*, pages 274–288. Springer.

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

[Leivant and Marion, 1997] Leivant, D. and Marion, J.-Y. (1997). 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.

[Longley, 2002] Longley, J. R. (2002). The sequentially realizable functionals. *apal*, 117:1–93.

[Mairson and Terui, 2003] Mairson, H. G. and Terui, K. (2003). On the computational complexity of cut-elimination in linear logic. In *ICTCS*, volume 2841 of *Lecture Notes in Computer Science*, pages 23–36. Springer.

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

[Milner, 1978] Milner, R. (1978). A theory of type polymorphism in programming languages. *Journal of Computer and System Science*, 17(3):348–375.

[Mitchell, 1984] Mitchell, J. C. (1984). Type inference and type containment. In *Proc. of the international symposium on Semantics of data types*, Lecture Notes in Computer Science, pages 257–277. Springer.

[Mitchell, 1988] Mitchell, J. C. (1988). Polymorphic type inference and containment. *Information and Computation*, 76(2/3):211–249. Reprinted in *Logical Foundations of Functional Programming* ed. G. Huet, Addison-Wesley (1990) 153–194.

[Odifreddi, 1999] Odifreddi, P. (1999). *Classical Recursion Theory.* North Holland, Amsterdam.

[Oitavem, 2001] 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*, volume 2183 of *Lecture Notes in Computer Science*, pages 170–190. Springer.

[Ong, 1995] Ong, C.-H. L. (1995). Correspondence between operational and denotational semantics. In Abramsky, S., Gabbay, D., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science, Vol 4*, pages 269–356. Oxford University Press.

[Paolini, 2006] Paolini, L. (2006). A stable programming language. *Information and Computation*, 204(3):339–375.

[Plotkin, 1977] Plotkin, G. D. (1977). LCF considerd as a programming language. *Theoretical Computer Science*, 5:225–255.

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

[Rémy, 2005] Rémy, D. (2005). Simple, partial type-inference for system F based on type-containment. In Danvy, O. and Pierce, B. C., editors, *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, ICFP 2005, Tallinn, Estonia, September 26-28, 2005*, pages 130–143. ACM.

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

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

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

[Scott, 1993] Scott, D. S. (1993). A type-theoretic alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440.

[Terui, 2000] Terui, K. (2000). Linear logical characterization of polyspace functions (extended abstract). Presented at the Workshop on Implicit Computational Complexity ICC'00, Santa Barbara, 2000 - Unpublished.

[Terui, 2001] Terui, K. (2001). Light affine lambda calculus and polytime strong normalization. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science* (LICS '01), pages 209–220. IEEE Computer Society.

[Terui, 2002] Terui, K. (2002). *Light logic and polynomial time computation.* PhD thesis, Keio University.

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

[Wells, 1994] Wells, J. B. (1994). Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science* (LICS '94), pages 176–185. IEEE Computer Society.

[Wells, 1999] Wells, J. B. (1999). Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156.

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Simone MARTINI, Professeur, Università di Bologna, Italy

Madame Jacqueline VAUZEILLE, Professeur, Université Paris 13, LIPN, Villetaneuse

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur GABOARDI Marco

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Linéarité : un outil analytique pour l'étude de la complexité et de la sémantique des langages de programmation"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Informatique** »

Fait à Vandoeuvre, le 04 décembre 2007
Le Président de l'I.N.P.L.,
F. LAURENT

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANŒUVRE CEDEX

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55